Министерство образования Российской Федерации

Пензенский государственный университет

**Кафедра «Вычислительная техника»**

**ОТЧЁТ**

По лабораторной работе №6

По дисциплине:

«Программирование на языке JAVA»

На тему:

«Сетевое взаимодействие в Java»

Выполнили:

студенты группы 20ВВП1

Верховский М. В.

Лукин В. Д.

Приняли:

к.т.н., доцент, Юрова О.В.

к.т.н., доцент, Карамышева Н.С.

**Пенза 2023**

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

**Задание на лабораторную работу:**

Модифицировать приложение из предыдущей лабораторной работы, реализовав клиент-серверную архитектуру, обеспечивающую распределенное вычисление определенного интеграла на сервере. При этом сервер использует 4 нити, как в предыдущей работе. Клиент не занимается вычислениями, а лишь реализует взаимодействие с пользователем и агрегацию результатов вычислений от клиентов. Протокол UDP.

**Ход выполнения лабораторной работы:**

**Листинг**

**Клиент:**

Table.java

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.\*;  
  
  
import static javax.swing.JOptionPane.ERROR\_MESSAGE;  
  
  
 public class Table implements Serializable {  
 private static LinkedList<RecIntegral> info = new LinkedList<>();  
 private LinkedList<String> values = new LinkedList<>();  
 private JButton AddButton;  
 private JButton DelButton;  
 private JButton CalcButton;  
 private JTextField Down;  
 private JTextField Up;  
 private JTextField Step;  
 public JTable MyTable;  
 public DefaultTableModel MyModel;  
 private JPanel MyPanel;  
 private JButton ClearButton;  
 private JButton StuffButton;  
 private JButton WriteButton;  
 private JButton DownloadButton;  
 private JButton WriteButtonSer;  
 private JButton DownloadButtonSer;  
  
 // Порт сервера, к которому собирается подключиться клиентский сокет  
 public final static int SERVICE\_PORT = 50001;  
 public Table() {  
  
 //Убираем возможность редактирования столбца с результатом  
 MyTable.setModel(new DefaultTableModel() {  
  
 @Override  
 public boolean isCellEditable(int rowIndex, int columnIndex) { // Возвращает true, если ячейка в rowIndex и columnIndex доступна для редактирования  
 return columnIndex != 3;  
 }  
 });  
  
 // Чтобы добавить и удалить строки из таблицы, нужно использовать DefaultTableModel  
 MyModel = (DefaultTableModel) MyTable.getModel();  
  
 // создание столбцов  
 MyModel.addColumn("Низ");  
 MyModel.addColumn("Верх");  
 MyModel.addColumn("Шаг");  
 MyModel.addColumn("Результат");  
  
 AddButton.addActionListener(new ActionListener() { // ActionListener хранит в себе метод, который активируется при нажатии кнопки  
 @Override  
 public void actionPerformed(ActionEvent e) { // actionPerformed - метод, который активируется при нажатии кнопки  
 //Обработка исключений с помощью блоков try-catch:  
 try {  
 if (Double.parseDouble(Step.getText()) == 0) {  
 throw new Exception1("Шаг");  
 }  
  
 } catch (Exception1 ex) {  
 ex.Message1(MyPanel, ERROR\_MESSAGE);  
 return;  
 }  
  
  
 try {  
 if (Double.parseDouble(Down.getText()) > 1000000 || Double.parseDouble(Down.getText()) < 0.000001) {  
 throw new Exception1("Нижний предел");  
 }  
 if (Double.parseDouble(Up.getText()) > 1000000 || Double.parseDouble(Up.getText()) < 0.000001) {  
 throw new Exception1("Верхний предел");  
 }  
 if (Double.parseDouble(Step.getText()) > 1000000 || Double.parseDouble(Step.getText()) < 0.000001) {  
 throw new Exception1("Шаг");  
 }  
  
 } catch (Exception1 ex) {  
 ex.Message2(MyPanel, ERROR\_MESSAGE);  
 return;  
 }  
  
  
 try {  
 if (Double.parseDouble(Down.getText()) > Double.parseDouble(Up.getText())) {  
 throw new Exception1("Пределы");  
 }  
  
 } catch (Exception1 ex) {  
 ex.Message3(MyPanel, ERROR\_MESSAGE);  
 return;  
 }  
  
  
 Double Up = Double.parseDouble(Table.this.Up.getText()); // getText возвращает текст, содержащийся в текстовом поле Up  
 Double Down = Double.parseDouble(Table.this.Down.getText());  
 Double Step = Double.parseDouble(Table.this.Step.getText());  
 MyModel.addRow(new Double[]{Down, Up, Step}); // addRow добавляет строку  
 info.add(new RecIntegral(Down, Up, Step, 0));//Работа с коллекцией  
  
 }  
 });  
  
 CalcButton.addActionListener(new ActionListener() { // ActionListener хранит в себе метод, который активируется при нажатии кнопки  
 @Override  
 public void actionPerformed(ActionEvent e) { // actionPerformed - метод, который активируется при нажатии кнопки  
  
 try {  
 //Создали экземпляр клиентского сокета  
 DatagramSocket clientSocket = new DatagramSocket();  
 //Получение IP-адреса сервера  
 InetAddress IPAddress = InetAddress.getByName("localhost");  
  
 //Создали соответствующие буферы  
 byte[] sendingDataBuffer = new byte[1024];  
 byte[] receivingDataBuffer = new byte[1024];  
  
 String data;  
 //UDP-пакеты  
 DatagramPacket sendingPacket;  
 DatagramPacket receivingPacket;  
  
  
 //записываем информацию из модела для передачи на сервер  
 for (int j = 0; j < 3; j++) {  
 data = new String(MyModel.getValueAt(0, j).toString());  
 sendingDataBuffer = data.getBytes();  
 //Создали UDP-пакет  
 sendingPacket = new DatagramPacket(sendingDataBuffer, sendingDataBuffer.length, IPAddress, SERVICE\_PORT);  
 //Отправили UDP-пакет серверу  
 clientSocket.send(sendingPacket);  
 }  
  
 //Получение обработанной инфы из сервера, занесение в таблицу  
 //Получили ответ от сервера  
 receivingPacket = new DatagramPacket(receivingDataBuffer, receivingDataBuffer.length);  
 clientSocket.receive(receivingPacket);  
 //Полученные данные занесли в таблицу  
 String receivedData = new String(receivingPacket.getData());  
 MyModel.setValueAt(Double.parseDouble(receivedData), 0, 3);

info.get(0).Res = Double.parseDouble(receivedData); //Работа с коллекцией

//Закрыли соединение с сервером через сокет  
 clientSocket.close();  
  
 }  
 catch (Exception ex) {  
 ex.printStackTrace();  
 }  
  
  
 }  
  
 });  
  
 DelButton.addActionListener(new ActionListener() { // ActionListener хранит в себе метод, который активируется при нажатии кнопки  
 @Override  
 public void actionPerformed(ActionEvent e) { // actionPerformed - метод, который активируется при нажатии кнопки  
 int indexRow = MyTable.getSelectedRow(); // Метод getSelectedRow() возвращает индекс первой выбранной строки  
 if (indexRow != -1) {  
 MyModel.removeRow(indexRow); // Метод removeRow() имеет параметр ind - индекс строки, которая будет удалена  
 info.remove(indexRow);//Работа с коллекцией  
 }  
 }  
 });  
  
 //Очистить таблицу  
 ClearButton.addActionListener(new ActionListener() { // ActionListener хранит в себе метод, который активируется при нажатии кнопки  
 @Override  
 public void actionPerformed(ActionEvent e) { // actionPerformed - метод, который активируется при нажатии кнопки  
 //Каждый раз, когда удаляем строку, количество строк будет меняться. Продолжаем цикл до тех пор, пока не останется строк  
 while (MyModel.getRowCount() > 0) {  
 MyModel.removeRow(0);  
 }  
 }  
 });  
  
 //Заполнить таблицу данными из коллекции  
 StuffButton.addActionListener(new ActionListener() { // ActionListener хранит в себе метод, который активируется при нажатии кнопки  
 @Override  
 public void actionPerformed(ActionEvent e) { // actionPerformed - метод, который активируется при нажатии кнопки  
 while (MyModel.getRowCount() > 0) {  
 MyModel.removeRow(0);  
 }  
 //Заполнение таблицы данными из коллекции  
 for (RecIntegral data : info) {  
 MyModel.addRow(new Object[]{data.Down1, data.Up1, data.Step1, data.Res});  
 }  
  
 }  
 });  
  
 //Загрузить в файл  
 WriteButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 JFileChooser window = new JFileChooser(); //Создание JFileChooser с указанием директории пользователя по умолчанию  
 window.setCurrentDirectory(new File(".")); //Установка директории по умолчанию  
 window.setDialogTitle("Загрузка в файл"); //Заголовок окна  
 window.setMultiSelectionEnabled(false); //Невозможность выбора сразу нескольких файлов  
 window.setFileSelectionMode(JFileChooser.FILES\_ONLY); //Выбор только файлов  
 window.setSelectedFile(new File("file1.txt")); //Выбор файла по умолчанию  
 window.showDialog(window, "Загрузить в файл"); //Открытие окна выбора файла с настроенным наименованием кнопки  
  
 File MyFile = window.getSelectedFile(); //Чтение выделенного файла  
  
 try {  
 FileWriter myWriter = new FileWriter(MyFile); //Создание объекта класса FileWriter для записи в файл на основе имени файла в файловой системе  
 BufferedWriter buf = new BufferedWriter(myWriter);  
  
 for (int i = 0; i < MyTable.getRowCount(); i++) {  
 for (int j = 0; j < MyTable.getColumnCount(); j++) {  
 buf.write(MyTable.getValueAt(i, j).toString() + " ");  
 }  
 buf.newLine(); //Метод newLine() использует собственное понятие разделителя строк платформы, определенное системным свойством line.separator  
 }  
  
 //Поток закрыт  
 buf.close();  
 myWriter.close();  
  
 } catch (IOException ex) {  
 throw new RuntimeException(ex);  
 }  
  
 }  
 });  
  
 //Загрузить из файла  
 DownloadButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 while (MyModel.getRowCount() > 0) {  
 MyModel.removeRow(0);  
 }  
  
 JFileChooser window = new JFileChooser(); //Создание JFileChooser с указанием директории пользователя по умолчанию  
 window.setCurrentDirectory(new File(".")); //Установка директории по умолчанию  
 window.setDialogTitle("Загрузка из файла"); //Заголовок окна  
 window.setMultiSelectionEnabled(false); //Невозможность выбора сразу нескольких файлов  
 window.setFileSelectionMode(JFileChooser.FILES\_ONLY); //Выбор только файлов  
 window.setSelectedFile(new File("file1.txt")); //Выбор файла по умолчанию  
 window.showDialog(window, "Загрузить из файла"); //Открытие окна выбора файла с настроенным наименованием кнопки  
  
 File MyFile = window.getSelectedFile(); //Чтение выделенного файла  
  
 try {  
 FileReader myReader = new FileReader(MyFile); //Создание объекта класса FileReader для чтения из файла на основе имени файла в файловой системе  
 BufferedReader buf = new BufferedReader(myReader); //BufferedReader записывает текст в поток вывода символов, буферизуя символы, чтобы обеспечить эффективную запись отдельных символов, массивов и строк  
  
 Object[] lines = buf.lines().toArray(); //Метод lines() — метод, который возвращает поток строк, извлеченных из заданной многострочной строки  
  
 for (int i = 0; i < lines.length; i++) {  
 String[] row = lines[i].toString().split(" "); //Метод split разделяет строку на подстроки, используя разделитель, который определяется с помощью регулярного выражения  
 MyModel.addRow(row);  
 }  
  
 //Поток закрыт  
 buf.close();  
 myReader.close();  
  
 } catch (IOException ex) {  
 throw new RuntimeException(ex);  
 }  
  
 }  
 });  
  
 //Сериализация  
 WriteButtonSer.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 JFileChooser window = new JFileChooser();  
 window.setCurrentDirectory(new File(".")); //Создание JFileChooser с указанием директории пользователя по умолчанию  
 window.setDialogTitle("Выберите файл для сериализации"); //Заголовок окна  
 window.setMultiSelectionEnabled(false); //Невозможность выбора сразу нескольких файлов  
 window.setFileSelectionMode(JFileChooser.FILES\_ONLY); //Выбор только файлов  
 window.setSelectedFile(new File("A.ser")); //Выбор файла по умолчанию  
 window.showDialog(window, "Выбрать"); //Открытие окна выбора файла с настроенным наименованием кнопки  
  
 File MyFile = window.getSelectedFile(); //Чтение выделенного файла  
  
 for (int i = 0; i < MyTable.getRowCount(); i++) {  
 for (int j = 0; j < MyTable.getColumnCount(); j++) {  
 values.add(String.valueOf(MyTable.getValueAt(i, j)));  
 }  
 }  
  
 try {  
 //создаем 2 потока для сериализации объекта и сохранения его в файл  
 FileOutputStream outputStream = new FileOutputStream(MyFile);  
 ObjectOutputStream out = new ObjectOutputStream(outputStream);  
 // сохраняем values в файл  
 out.writeObject(values);  
 //закрываем поток и освобождаем ресурсы  
 outputStream.close();  
 out.close();  
  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
 });  
  
 //Десериализация  
 DownloadButtonSer.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 while (MyModel.getRowCount() > 0) {  
 MyModel.removeRow(0);  
 }  
  
 JFileChooser window = new JFileChooser(); //Создание JFileChooser с указанием директории пользователя по умолчанию  
 window.setCurrentDirectory(new File(".")); //Установка директории по умолчанию  
 window.setDialogTitle("Выберите файл для десериализации"); //Заголовок окна  
 window.setMultiSelectionEnabled(false); //Невозможность выбора сразу нескольких файлов  
 window.setFileSelectionMode(JFileChooser.FILES\_ONLY); //Выбор только файлов  
 window.setSelectedFile(new File("A.ser")); //Выбор файла по умолчанию  
 window.showDialog(window, "Выбрать"); //Открытие окна выбора файла с настроенным наименованием кнопки  
  
 File MyFile = window.getSelectedFile(); //Чтение выделенного файла  
  
 try {  
 //создаем 2 потока для десериализации объекта и выгрузки его из файла  
 FileInputStream inputStream = new FileInputStream(MyFile);  
 ObjectInputStream in = new ObjectInputStream(inputStream);  
  
 values = (LinkedList<String>) in.readObject(); //Метод readObject для чтения объекта из потока  
 //закрываем поток и освобождаем ресурсы  
 in.close();  
 inputStream.close();  
  
 } catch (IOException | ClassNotFoundException ex) {  
 ex.printStackTrace();  
 }  
  
 int i = 0;  
 while (i < values.size()) {  
 MyModel.addRow(new Object[]{  
 values.get(i),  
 values.get(i + 1),  
 values.get(i + 2),  
 values.get(i + 3)  
 });  
 i = i + 4;  
 }  
 }  
 });  
  
 }  
  
  
 public static void main(String[] args) {  
 JFrame frame = new JFrame("Интегральная гипербола кибернетики"); // создаём окно с рамкой  
 frame.setVisible(true); // показывает или скрывает это окно в зависимости от значения параметра  
 frame.setContentPane(new Table().MyPanel);//задаёт содержимое окна с рамкой  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);//Устанавливает операцию при закрытии окна закончить работу приложения,  
 // которая будет выполняться по умолчанию, когда пользователь инициирует «закрытие» этого окна  
 frame.setSize(420,600); // размер окна  
 frame.setLocation(120,100); // расположение окна  
  
 }  
  
}

RecIntegral.java

import java.io.Serializable;  
  
public class RecIntegral implements Serializable {  
  
 public double Up1;  
 public double Down1;  
 public double Step1;  
 public double Res;  
  
 public RecIntegral(double Down1, double Up1, double Step1, double Res){  
 this.Down1 = Down1;  
 this.Up1 = Up1;  
 this.Step1 = Step1;  
 this.Res = Res;  
  
 }  
  
}

Exception1.java

import java.awt.\*;  
import static javax.swing.JOptionPane.showMessageDialog;  
  
//класс Exception1, унаследованный от класса Exception:  
public class Exception1 extends Exception{  
  
 private String title;  
  
 public Exception1(String title) {  
 this.title = title;  
 }  
  
 //Диалоговое окно с заголовком, cообщением и типом сообщения  
 public void Message1(Component parent, int optionType) {  
 showMessageDialog(parent, "Введён нулевой шаг", title, optionType);  
 }  
 public void Message2(Component parent, int optionType) {  
 showMessageDialog(parent, "Выход за пределы диапазона (0.000001; 1000000)", title, optionType);  
 }  
  
 public void Message3(Component parent, int optionType) {  
 showMessageDialog(parent, "Верхний предел не может быть меньше нижнего предела", title, optionType);  
 }  
  
}

**Сервер:**

Server.java

import java.io.\*;  
import java.net.\*;  
import java.util.concurrent.Callable;  
import java.util.concurrent.ExecutionException;  
import java.util.concurrent.FutureTask;  
  
public class Server {  
 //Серверный UDP-сокет запущен на этом порту  
 public final static int SERVICE\_PORT = 50001;  
 public static void main (String[] args){  
  
 try {  
 //Создали новый экземпляр DatagramSocket, чтобы получать ответы от клиента  
 DatagramSocket serverSocket = new DatagramSocket(SERVICE\_PORT);  
  
 //Создайте буферы для хранения отправляемых и получаемых данных  
 byte[] receivingDataBuffer = new byte[1024];  
 byte[] sendingDataBuffer = new byte[1024];  
  
 //Создали экземпляр UDP-пакета для хранения клиентских данных с использованием буфера для полученных данных  
 DatagramPacket inputPacket = new DatagramPacket(receivingDataBuffer, receivingDataBuffer.length);  
  
 String receivedData; //данные клиента  
 double[] func = new double[3]; //массив для хранения данных таблицы (min, max, step)  
  
 String SendingMessage; //результат  
 InetAddress senderAddress; //IP-адрес клиента  
 int senderPort; //порт клиента  
 DatagramPacket outputPacket; //UDP-пакет с данными для отправки клиенту  
  
  
 double area = 0;  
 for (int j = 0; j < 3; j++) {  
 //Получение данных от клиента и сохранение их в inputPacket  
 serverSocket.receive(inputPacket);  
 //Занесение отправленных клиентом данных в массив func  
 receivedData = new String(inputPacket.getData());  
 func[j] = Double.parseDouble(receivedData);  
 }  
 double a = func[0];  
 double b = func[1];  
 double h = func[2];  
  
 System.out.println("min (a) = " + a);  
 System.out.println("max (b) = " + b);  
 System.out.println("step (h) = " + h);  
  
 Callable<Double> t1 = new Callable4(a, a + ((b - a) \* 0.25), h);  
 Callable<Double> t2 = new Callable4(a + ((b - a) \* 0.25), a + ((b - a) \* 0.5), h);  
 Callable<Double> t3 = new Callable4(a + ((b - a) \* 0.5), a + ((b - a) \* 0.75), h);  
 Callable<Double> t4 = new Callable4(a + ((b - a) \* 0.75), b, h);  
  
 FutureTask futureTask1 = new FutureTask(t1);  
 FutureTask futureTask2 = new FutureTask(t2);  
 FutureTask futureTask3 = new FutureTask(t3);  
 FutureTask futureTask4 = new FutureTask(t4);  
  
 new Thread(futureTask1).start();  
 new Thread(futureTask2).start();  
 new Thread(futureTask3).start();  
 new Thread(futureTask4).start();  
  
 try {  
 System.out.println("Интервал 1: " + futureTask1.get());  
 System.out.println("Интервал 2: " + futureTask2.get());  
 System.out.println("Интервал 3: " + futureTask3.get());  
 System.out.println("Интервал 4: " + futureTask4.get());  
 area = (double) futureTask1.get() + (double) futureTask2.get() + (double) futureTask3.get() + (double) futureTask4.get();  
 System.out.println("Итог: " + area);  
  
  
 //отправка на клиент результатов  
 SendingMessage = Double.toString(area); //перевод результата (area) в String  
 sendingDataBuffer = SendingMessage.getBytes(); //результат преобразуем в байты  
 senderAddress = inputPacket.getAddress(); //Получение IP-адреса клиента  
 senderPort = inputPacket.getPort(); ///Получение порта клиента  
 //Создание нового UDP-пакета с данными, чтобы отправить их клиенту  
 outputPacket = new DatagramPacket(sendingDataBuffer, sendingDataBuffer.length, senderAddress, senderPort);  
 serverSocket.send(outputPacket); //Отправка пакета (результата) клиенту  
  
 } catch (InterruptedException | ExecutionException ex) {  
 throw new RuntimeException(ex);  
 }  
 //Закрыли соединение сокетов  
 serverSocket.close();  
 }  
 catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
}

Callable4.java

import java.util.concurrent.Callable;  
  
class Callable4 implements Callable<Double> {  
 volatile double a;  
 volatile double b;  
 volatile double h;  
 volatile double area = 0.0;  
  
 public Callable4(double a, double b, double h) {  
 this.a = a;  
 this.b = b;  
 this.h = h;  
 }  
  
 public synchronized Double call() {  
  
 for (int i = 0; i < (b - a) / h; i++) {  
  
 if (a + (i + 1) \* h <= b) {  
 area += h \* (0.5 \* (Math.tan(a + i \* h) + Math.tan(a + (i + 1) \* h)));  
 } else {  
 area += h \* (0.5 \* (Math.tan(a + i \* h) + Math.tan(b)));  
 }  
  
 }  
 return area;  
 }  
  
}

**Результаты выполнения программы**

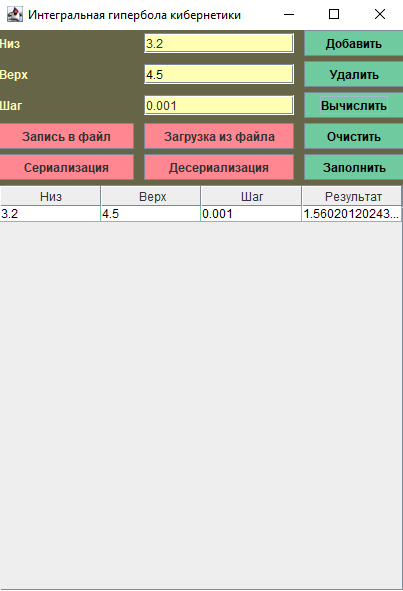


Рисунок №1 – Запуск клиент-серверной архитектуры, обеспечивающей распределенное вычисление определенного интеграла

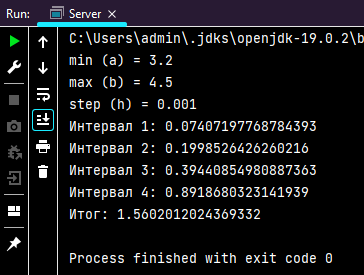


Рисунок №2 – Распределенное вычисление определенного интеграла выполняется на сервере. При этом сервер использует 4 нити, как в предыдущей работе.

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