Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Программирование на языке JAVA»

Выполнили:

студенты группы 20ВВП2

Киреев Б.П.

Барсуков Н.И.

Проверил:

Юрова О.В.

Карамышева Н.С.

Пенза 2023

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

**Вариант №6**

**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:**

Main.java

public class Main {  
 public static void main(String[] args) {  
 new MainForm();  
 }  
}

MainForm.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.Socket;  
import java.util.LinkedList;  
import java.util.Scanner;  
import java.util.Vector;  
import java.util.concurrent.Callable;  
import java.util.concurrent.ExecutionException;  
import java.util.concurrent.FutureTask;  
  
public class MainForm extends JFrame {  
 private JTextField TextField\_UpperLimit;  
 private JTextField TextField\_LowerLimit;  
 private JPanel WorkPlace;  
 private JButton addButton;  
 private JButton deleteButton;  
 private JTable table1;  
 private JButton calculateButton;  
 private JLabel JLabel\_UpperLimit;  
 private JLabel JLabel\_LowerLimit;  
 private JLabel JLabel\_Step;  
 private JTextField TextField\_Step;  
 private JButton fillButton;  
 private JButton clearButton;  
 private JButton saveButton;  
 private JButton loadButton;  
 private JButton serializationButton;  
 private JButton deserializationButton;  
 private DefaultTableModel model;  
  
 public LinkedList<RecIntegral> list = new LinkedList<>();  
  
  
  
 public MainForm(){  
 model = (DefaultTableModel)table1.getModel();  
 model.addColumn("Верхний предел");  
 model.addColumn("Нижний предел");  
 model.addColumn("Шаг");  
 model.addColumn("Результат");  
 table1.setModel(model);  
  
 addButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 try {  
 if (Double.*parseDouble*(TextField\_UpperLimit.getText()) > 1000000 ||  
 Double.*parseDouble*(TextField\_UpperLimit.getText()) < 0.000001)  
 {  
 throw new MyException("Выход за пределы диапазона", "Верхний лимит");  
 }  
  
 if (Double.*parseDouble*(TextField\_LowerLimit.getText()) > 1000000 ||  
 Double.*parseDouble*(TextField\_LowerLimit.getText()) < 0.000001)  
 {  
 throw new MyException("Выход за пределы диапазона", "Нижний лимит");  
 }  
  
 if (Double.*parseDouble*(TextField\_Step.getText()) > 1000000 ||  
 Double.*parseDouble*(TextField\_Step.getText()) < 0.000001)  
 {  
 throw new MyException("Выход за пределы диапазона", "Шаг");  
 }  
  
 if (Double.*parseDouble*(TextField\_Step.getText()) == 0)  
 {  
 throw new MyException("Шаг не может быть равен нулю", "Шаг");  
 }  
 } catch (MyException ex)  
 {  
 ex.ShowMessage(WorkPlace);  
 return;  
 }  
  
 model.addRow(new Object[]{  
 Double.*parseDouble*(TextField\_UpperLimit.getText()),  
 Double.*parseDouble*(TextField\_LowerLimit.getText()),  
 Double.*parseDouble*(TextField\_Step.getText()),  
 0  
 });  
 list.add(new RecIntegral(Double.*parseDouble*(TextField\_UpperLimit.getText()),  
 Double.*parseDouble*(TextField\_LowerLimit.getText()),  
 Double.*parseDouble*(TextField\_Step.getText()),  
 0.0));  
  
 //TextField\_UpperLimit.setText("");  
 //TextField\_LowerLimit.setText("");  
 //TextField\_Step.setText("");  
 }  
 });  
  
 deleteButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (table1.getSelectedRow() != -1){  
 list.remove(table1.getSelectedRow());  
 model.removeRow(table1.getSelectedRow());  
 }  
 }  
 });  
  
 calculateButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 try {  
 Socket client = new Socket("localhost", 8081);  
  
 DataInputStream input\_ = new DataInputStream(client.getInputStream());  
 DataOutputStream output\_ = new DataOutputStream(client.getOutputStream());  
  
 for (int j = 0; j < 6; j++) {  
 double downLimit = Double.*parseDouble*(String.*valueOf*(model.getValueAt(j, 0)));  
 double upLimit = Double.*parseDouble*(String.*valueOf*(model.getValueAt(j, 1)));  
 double step = Double.*parseDouble*(String.*valueOf*(model.getValueAt(j, 2)));  
  
 output\_.writeDouble(downLimit);  
 output\_.writeDouble(upLimit);  
 output\_.writeDouble(step);  
  
 Double result = input\_.readDouble();  
 System.*out*.println(result);  
  
 model.setValueAt(result, j, 3);  
 list.add(new RecIntegral(downLimit, upLimit, step, result));  
 }  
 client.close();  
// String nums = "";  
// for (int i = 0; i < 6; i++){  
// nums += model.getValueAt(i, 0).toString() + ",";  
// nums += model.getValueAt(i, 1).toString() + ",";  
// nums += model.getValueAt(i, 2).toString() + ",";  
// }  
//  
// Scanner input = new Scanner(client.getInputStream());  
// PrintWriter output = new PrintWriter(client.getOutputStream());  
// output.println(nums);  
// output.flush();  
//  
// String str = input.nextLine();  
// System.out.println(str);  
// String[] res = str.split(",");  
//  
// for (int i = 0; i < model.getRowCount(); i++) {  
// model.setValueAt(Double.valueOf(res[i]), i, 3);  
// }  
 } catch (IOException ex) {  
 System.*out*.println(ex);  
 }  
  
 // table1.repaint();  
  
// for (int i = 0; i < 6; i++) //просчёт первых 6-и строк шестью потоками  
// {  
// Vector data = model.getDataVector().get(i);  
//  
// Double upper = (Double)data.get(0);  
// Double low = (Double)data.get(1);  
// Double step = (Double)data.get(2);  
// Double result;  
//  
//  
// Callable<Double> thread = new MyThread(low, upper, step);  
// FutureTask futureTask = new FutureTask(thread);  
// new Thread(futureTask).start();  
//  
// try {  
// result = (double) futureTask.get();  
// System.out.println("Поток " + i + " Результат: " + result);  
//  
// } catch (InterruptedException | ExecutionException ex) {  
// throw new RuntimeException(ex);  
// }  
//  
// data.set(3, result);  
// list.get(i).res = result;  
// }  
//  
// table1.repaint();  
 }  
 });  
  
 setContentPane(WorkPlace);  
 setSize(1000, 600);  
 setVisible(true);  
 setDefaultCloseOperation(WindowConstants.*EXIT\_ON\_CLOSE*);  
  
 fillButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 for (RecIntegral memList :  
 list) {  
 model.addRow(new Object[]{  
 memList.upLim,  
 memList.lowLim,  
 memList.st,  
 memList.res  
 });  
 }  
 }  
 });  
 clearButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int modelSize = model.getRowCount() - 1;  
 for (int i = modelSize; i >= 0; i--)  
 {  
 model.removeRow(i);  
 }  
 }  
 });  
 saveButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 File file = new File("Data");  
 PrintWriter pw = null;  
 try {  
 pw = new PrintWriter(file);  
 } catch (FileNotFoundException ex) {  
 throw new RuntimeException(ex);  
 }  
 for (RecIntegral itVar : list)  
 {  
 pw.print(itVar.upLim.toString() + " ");  
 pw.print(itVar.lowLim.toString() + " ");  
 pw.print(itVar.st.toString() + " ");  
 pw.print(itVar.res.toString() + " ");  
 pw.print("\n");  
 }  
 pw.close();  
  
 }  
 });  
 loadButton.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 из файла"); //Заголовок окна  
 window.setMultiSelectionEnabled(false); //Невозможность выбора сразу нескольких файлов  
 window.setFileSelectionMode(JFileChooser.*FILES\_ONLY*); //Выбор только файлов  
 window.setSelectedFile(new File("Data")); //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  
  
 int modelSize = model.getRowCount() - 1;  
 for (int i = modelSize; i >= 0; i--)  
 {  
 model.removeRow(i);  
 }  
  
 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  
 model.addRow(new Object[]{Double.*parseDouble*(row[0]), Double.*parseDouble*(row[1]), Double.*parseDouble*(row[2]), Double.*parseDouble*(row[3])});  
 }  
  
 //Поток закрыт  
 buf.close();  
 myReader.close();  
  
 } catch (IOException ex) {  
 throw new RuntimeException(ex);  
 }  
 }  
 });  
 serializationButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("RecInt.dat"));  
 oos.writeObject(list);  
 oos.close();  
 } catch (IOException ex) {  
 System.*out*.println(ex.getMessage());  
 }  
 }  
 });  
 deserializationButton.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 из файла"); //Заголовок окна  
 window.setMultiSelectionEnabled(false); //Невозможность выбора сразу нескольких файлов  
 window.setFileSelectionMode(JFileChooser.*FILES\_ONLY*); //Выбор только файлов  
 window.setSelectedFile(new File("RecInt.dat")); //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 (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(MyFile.getName()))) {  
 list.clear();  
 list = ((LinkedList<RecIntegral>)ois.readObject());//бла бла бла  
 } catch (Exception ex) {  
 System.*out*.println(ex.getMessage());  
 }  
 }  
 });  
 }  
  
 private void createUIComponents() {  
 table1 = new JTable() {  
 @Override  
 public boolean isCellEditable(int row, int col) {  
 return (col != 3);  
 }  
 };  
 }  
  
};

MyThread.java

import java.util.concurrent.Callable;  
  
public class MyThread implements Callable<Double> {  
 public Double upper;  
 public Double low;  
 public Double step;  
 public Double result = 0.0;  
  
 public MyThread(double low, double upper, double step) {  
 this.low = low;  
 this.upper = upper;  
 this.step = step;  
  
 }  
  
 public Double call() {  
  
 {  
 for (int i = 0; i < (upper - low) / step; i++) {  
 if (low + (i + 1) \* step < upper)  
 result += 0.5 \* step \* (Math.*sqrt*(low + i \* step) + Math.*sqrt*(low + (i + 1) \* step));  
 else  
 result += 0.5 \* step \* (Math.*sqrt*(low + i \* step) + Math.*sqrt*(upper));  
 }  
 return result;  
 }  
  
  
 }  
  
}

RecIntegral.java

import java.io.Serializable;  
  
public class RecIntegral implements Serializable {  
 public Double upLim;  
 public Double lowLim;  
 public Double st;  
 public Double res;  
 public int Size = 4;  
  
 public RecIntegral(Double upLim, Double lowLim, Double st, Double res) {  
 this.upLim = upLim;  
 this.lowLim = lowLim;  
 this.st = st;  
 this.res = res;  
 }  
}

MyException.java

import javax.swing.\*;  
import java.awt.\*;  
  
public class MyException extends Exception {  
 private String message;  
 private String title;  
  
 public MyException(String message, String title) {  
 this.message = message;  
 this.title = title;  
 }  
  
 public void ShowMessage(Component parent)  
 {  
 JOptionPane.*showMessageDialog*(parent, message, title, JOptionPane.*ERROR\_MESSAGE*);  
 }  
}

**Листинг сервера**

Main.java

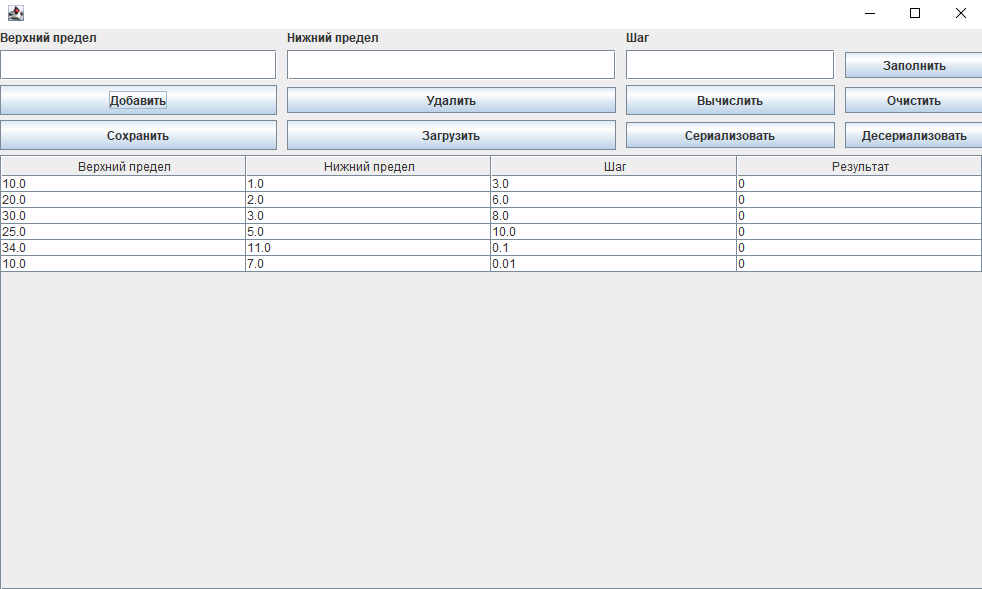
import java.io.DataInputStream;  
import java.io.DataOutputStream;  
import java.io.IOException;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.Arrays;  
import java.util.Scanner;  
import java.util.concurrent.Callable;  
import java.util.concurrent.ExecutionException;  
import java.util.concurrent.FutureTask;  
  
public class Main {  
  
 public static void main(String[] args) throws IOException {  
  
 //Double[] result = new Double[6];  
 ServerSocket server = new ServerSocket(8081);  
  
 Socket client = server.accept();  
  
 DataInputStream input\_ = new DataInputStream(client.getInputStream());  
 DataOutputStream output\_ = new DataOutputStream(client.getOutputStream());  
  
 for (int i = 0; i < 6; i++) {  
 Double down = input\_.readDouble();  
 System.*out*.println(down);  
 Double up = input\_.readDouble();  
 System.*out*.println(up);  
 Double step = input\_.readDouble();  
 System.*out*.println(step);  
 Double result;  
  
 Callable<Double> calc = new MainThread(down, up, step);  
 FutureTask futureTask = new FutureTask(calc);  
 new Thread(futureTask).start();  
  
  
 try {  
 result = (Double)futureTask.get();  
 System.*out*.println(result);  
  
 } catch (ExecutionException | InterruptedException ex) {  
 throw new RuntimeException(ex);  
 }  
  
 output\_.writeDouble(result);  
  
 }  
  
 server.close();  
   
 }  
}

MainThread.java

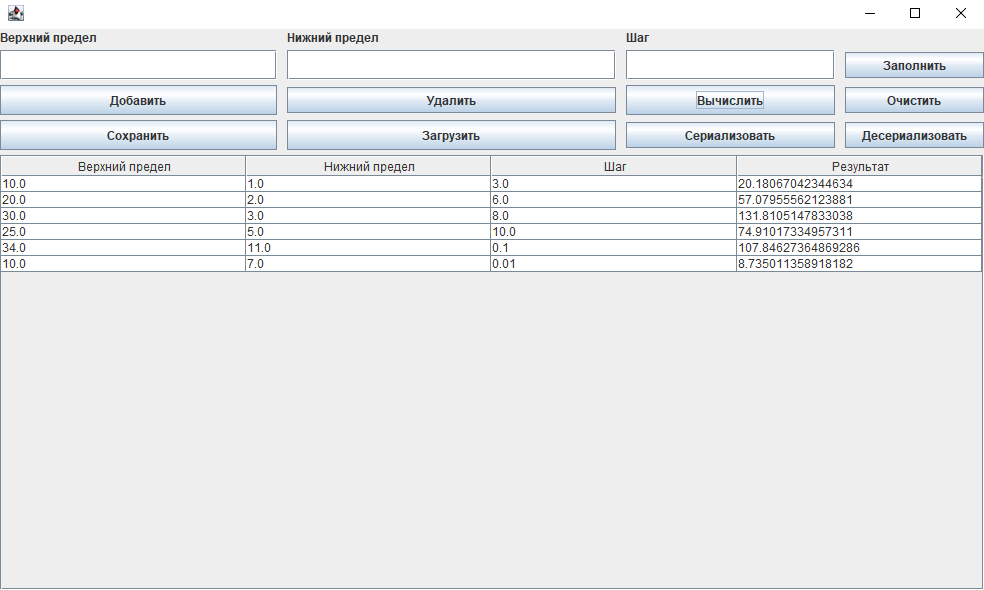
import java.util.concurrent.Callable;  
  
public class MainThread implements Callable<Double> {  
 public Double upper;  
 public Double low;  
 public Double step;  
 public Double result = 0.0;  
  
 public MainThread(Double low, Double upper, Double step) {  
 this.low = low;  
 this.upper = upper;  
 this.step = step;  
  
 }  
  
 public Double call() {  
 for (int i = 0; i < (upper - low) / step; i++) {  
 if (low + (i + 1) \* step < upper)  
 result += 0.5 \* step \* (Math.*sqrt*(low + i \* step) + Math.*sqrt*(low + (i + 1) \* step));  
 else  
 result += 0.5 \* step \* (Math.*sqrt*(low + i \* step) + Math.*sqrt*(upper));  
 }  
 return result;  
 }  
  
}

**Результат работы:**

Ввод данных:



Результат:



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