Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по дисциплине «Программирование на языке Java»

на тему «Сетевое взаимодействие в Java»

Выполнили: студенты группы 22ВВП1

Беляев Д. И.

Демин М. С.

Приняли:

Юрова О. В.

Карамышева Н. С.

Пенза 2025

**Название**

Сетевое взаимодействие в Java

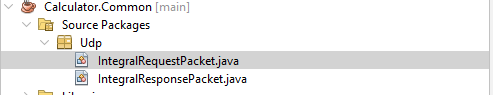
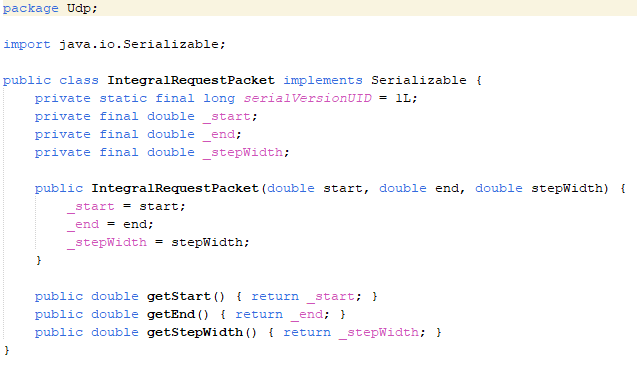
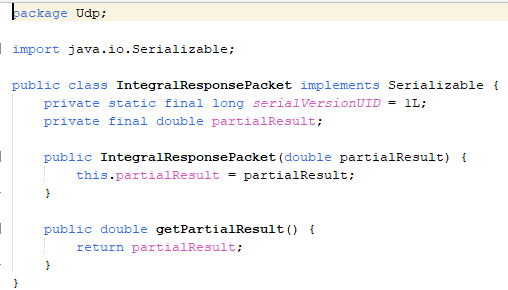
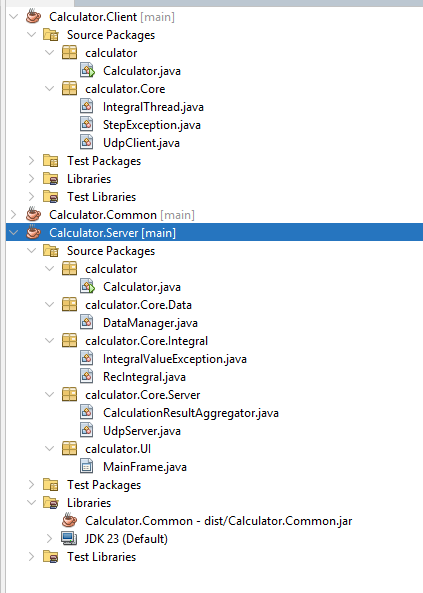
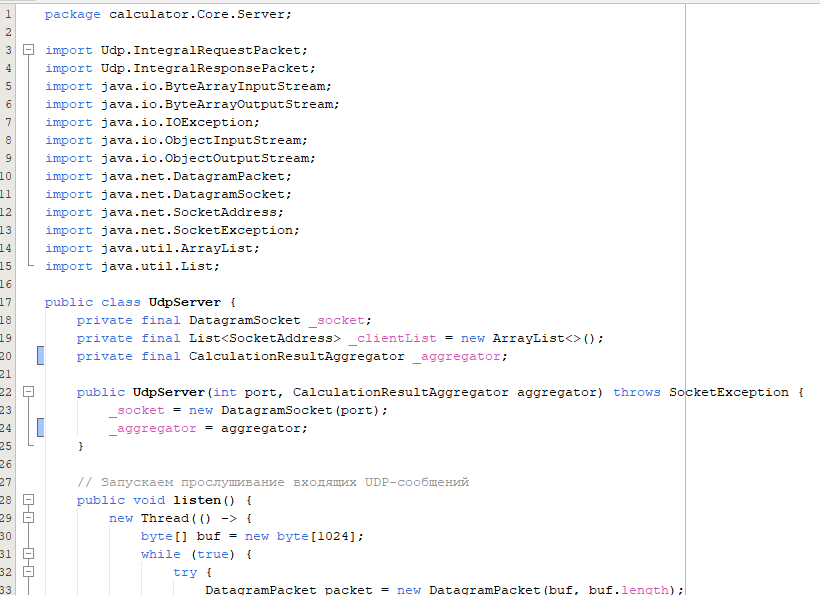
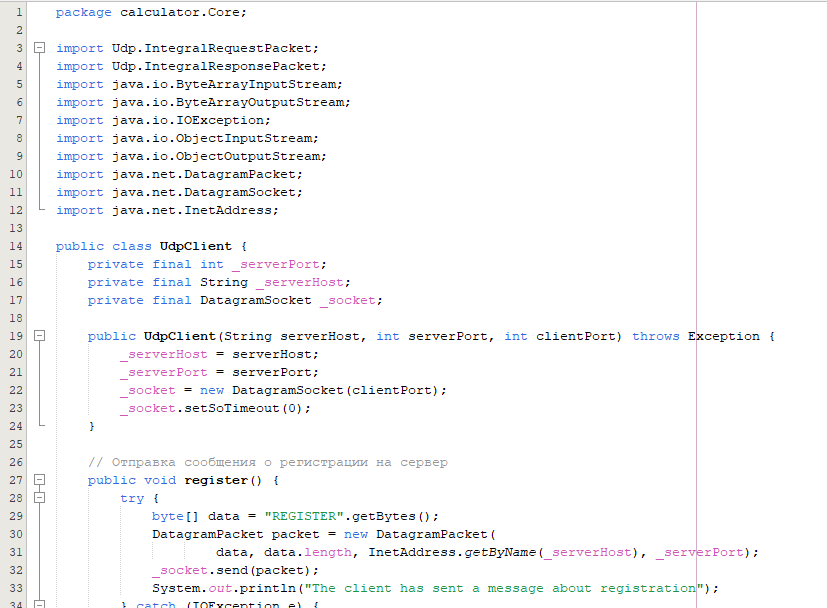
**Цель работы**

Научиться создавать клиент-серверные приложения c использованием стандартных классов Java

**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.

**Ход работы**

1. Сделали проект, который отвечает за классы, которые передаются между клиентом и сервером  
   ****  
   ****  
   ****
2. Разделили проект на клиент и сервер  
     
   ****

**Листинг**

package calculator;

import calculator.Core.UdpClient;

public class Calculator {

public static void main(String[] args) {

try {

String serverHost = "localhost";

int serverPort = 5000;

int clientPort = 0; // уникальный порт для этого клиента

UdpClient client = new UdpClient(serverHost, serverPort, clientPort);

client.register();

client.listenForRequests();

} catch (Exception e) {

e.printStackTrace();

}

}

}  
  
package calculator.Core;

public class IntegralThread extends Thread {

private final double \_start;

private final double \_end;

private final double \_stepWidth;

private double \_partialResult = 0.0;

public IntegralThread(double start, double end, double stepWidth) {

\_start = start;

\_end = end;

\_stepWidth = stepWidth;

}

public double getPartialResult() {

return \_partialResult;

}

@Override

public void run() {

System.out.println("Start calculate: [" +

\_start + ", " +

\_end + ", " +

\_stepWidth + "]...");

for (double x = \_start; x < \_end; x += \_stepWidth) {

double nextX = Math.min(x + \_stepWidth, \_end);

double area = (nextX - x) \* (Math.exp(-x) + Math.exp(-nextX)) / 2.0;

\_partialResult += area;

}

System.out.println("End calculate: [" +

\_start + ", " +

\_end + ", " +

\_stepWidth + "]");

}

}  
  
package calculator.Core;

public class StepException extends Exception {

public StepException(double step, double lowerBound, double upperBound){

String message = String.format(

"Step exception (Step is bigger than borders): step - %f, lower bound - %f, upper bound - %f",

step, lowerBound, upperBound);

super(message);

}

public String getExceptionName(){

return "StepException";

}

}  
  
package calculator.Core;

import Udp.IntegralRequestPacket;

import Udp.IntegralResponsePacket;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

public class UdpClient {

private final int \_serverPort;

private final String \_serverHost;

private final DatagramSocket \_socket;

public UdpClient(String serverHost, int serverPort, int clientPort) throws Exception {

\_serverHost = serverHost;

\_serverPort = serverPort;

\_socket = new DatagramSocket(clientPort);

\_socket.setSoTimeout(0);

}

// Отправка сообщения о регистрации на сервер

public void register() {

try {

byte[] data = "REGISTER".getBytes();

DatagramPacket packet = new DatagramPacket(

data, data.length, InetAddress.getByName(\_serverHost), \_serverPort);

\_socket.send(packet);

System.out.println("The client has sent a message about registration");

} catch (IOException e) {

e.printStackTrace();

}

}

public void close() {

if (\_socket != null && !\_socket.isClosed()) {

\_socket.close();

}

}

// Прослушивание входящих UDP-запросов

public void listenForRequests() {

new Thread(() -> {

byte[] buf = new byte[1024];

while (true) {

try {

DatagramPacket packet = new DatagramPacket(buf, buf.length);

\_socket.receive(packet);

IntegralRequestPacket request = deserializeRequest(packet);

System.out.println("Request received: [" +

request.getStart() + ", " +

request.getEnd() + ", " +

request.getStepWidth() + "]");

double partialResult = processRequest(request);

sendResponse(packet.getAddress(), packet.getPort(), partialResult);

System.out.println("The result has been sent: " + partialResult);

} catch (Exception e) {

e.printStackTrace();

}

}

}).start();

}

// Преобразование UDP-пакета в объект запроса

private IntegralRequestPacket deserializeRequest(DatagramPacket packet) throws Exception {

ByteArrayInputStream bais = new ByteArrayInputStream(packet.getData(), 0, packet.getLength());

ObjectInputStream ois = new ObjectInputStream(bais);

return (IntegralRequestPacket) ois.readObject();

}

// Обработка запроса с использованием вычислительного класса

private double processRequest(IntegralRequestPacket request) throws StepException {

final int THREAD\_COUNT = 5;

double start = request.getStart();

double end = request.getEnd();

double stepWidth = request.getStepWidth();

double sign = 1.0;

if (start > end) {

double tmp = start;

start = end;

end = tmp;

sign = -1.0;

}

if (stepWidth > (end - start)) {

throw new StepException(stepWidth, start, end);

}

double totalInterval = end - start;

double subIntervalLength = totalInterval / THREAD\_COUNT;

IntegralThread[] threads = new IntegralThread[THREAD\_COUNT];

for (int i = 0; i < THREAD\_COUNT; i++) {

double subStart = start + i \* subIntervalLength;

double subEnd = (i == THREAD\_COUNT - 1) ? end : subStart + subIntervalLength;

threads[i] = new IntegralThread(subStart, subEnd, stepWidth);

threads[i].start();

}

double sum = 0.0;

for (int i = 0; i < THREAD\_COUNT; i++) {

try {

threads[i].join();

sum += threads[i].getPartialResult();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

return sign \* sum;

}

// Отправка результата вычисления обратно серверу

private void sendResponse(InetAddress address, int port, double partialResult) throws Exception {

IntegralResponsePacket response = new IntegralResponsePacket(partialResult);

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ObjectOutputStream oos = new ObjectOutputStream(baos);

oos.writeObject(response);

oos.flush();

byte[] responseData = baos.toByteArray();

DatagramPacket responsePacket = new DatagramPacket(responseData, responseData.length, address, port);

\_socket.send(responsePacket);

}

}  
  
package Udp;

import java.io.Serializable;

public class IntegralRequestPacket implements Serializable {

private static final long serialVersionUID = 1L;

private final double \_start;

private final double \_end;

private final double \_stepWidth;

public IntegralRequestPacket(double start, double end, double stepWidth) {

\_start = start;

\_end = end;

\_stepWidth = stepWidth;

}

public double getStart() { return \_start; }

public double getEnd() { return \_end; }

public double getStepWidth() { return \_stepWidth; }

}

package Udp;

import java.io.Serializable;

public class IntegralResponsePacket implements Serializable {

private static final long serialVersionUID = 1L;

private final double partialResult;

public IntegralResponsePacket(double partialResult) {

this.partialResult = partialResult;

}

public double getPartialResult() {

return partialResult;

}

}  
  
package calculator;

import calculator.Core.Server.CalculationResultAggregator;

import calculator.Core.Server.UdpServer;

import calculator.UI.MainFrame;

import java.net.SocketException;

import javax.swing.SwingUtilities;

public class Calculator {

public static void main(String[] args) {

try {

CalculationResultAggregator aggregator = new CalculationResultAggregator();

UdpServer udpServer = new UdpServer(5000, aggregator);

udpServer.listen();

System.out.println("The UDP server is running on port 5000...");

SwingUtilities.invokeLater(() -> {

MainFrame frame = new MainFrame(udpServer, aggregator);

frame.setVisible(true);

});

} catch (SocketException e) {

e.printStackTrace();

}

}

}  
  
package calculator.Core.Data;

import calculator.Core.Integral.RecIntegral;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.util.ArrayList;

import java.util.List;

public class DataManager {

// Save data as text

public static void saveDataAsText(File file, List<RecIntegral> integrals) throws IOException {

try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {

for (RecIntegral integral : integrals) {

writer.write(integral.getTopBorder() + ";"

+ integral.getBottomBorder() + ";"

+ integral.getStepWidth() + ";"

+ integral.getResult());

writer.newLine();

}

}

}

// Load data from text

public static ArrayList<RecIntegral> loadDataFromText(File file) throws IOException {

ArrayList<RecIntegral> integrals = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader(file))) {

String line;

while ((line = reader.readLine()) != null) {

String[] parts = line.split(";");

if (parts.length < 4) {

continue;

}

double top = Double.parseDouble(parts[0]);

double bottom = Double.parseDouble(parts[1]);

double step = Double.parseDouble(parts[2]);

double result = Double.parseDouble(parts[3]);

RecIntegral integral = new RecIntegral(top, bottom, step, result);

integrals.add(integral);

}

} catch (Exception e) {

throw new IOException("Error when load data: " + e.getMessage());

}

return integrals;

}

// Save data as binary

public static void saveDataAsBinary(File file, List<RecIntegral> integrals) throws IOException {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(file))) {

oos.writeObject(integrals);

}

}

// Load data from binary

public static ArrayList<RecIntegral> loadDataFromBinary(File file) throws IOException, ClassNotFoundException {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {

return (ArrayList<RecIntegral>) ois.readObject();

}

}

}  
  
package calculator.Core.Integral;

public class IntegralValueException extends Exception {

public IntegralValueException(double lowerBound, double upperBound){

String message = String.format(

"Value exception: lower bound - %f, upper bound - %f",

lowerBound, upperBound);

super(message);

}

public String getExceptionName(){

return "IntegralValueException";

}

}

package calculator.Core.Integral;

import java.io.Serializable;

public class RecIntegral implements Serializable {

private static final long serialVersionUID = 1L;

private static final double MINIMAL\_DOUBLE\_VALUE = 0.000001;

private static final double MAXIMUM\_DOUBLE\_VALUE = 1000000;

private double \_topBorder = 0.0;

private double \_bottomBorder = 0.0;

private double \_stepWidth = 0.0;

private double \_result = 0.0;

public RecIntegral(double topBorder, double bottomBorder, double stepWidth) throws IntegralValueException {

this(topBorder, bottomBorder, stepWidth, Double.NaN);

}

public RecIntegral(

double topBorder,

double bottomBorder,

double stepWidth,

double previewResult)

throws IntegralValueException {

\_topBorder = topBorder;

\_bottomBorder = bottomBorder;

// Проверка значений на допустимый диапазон

if (bottomBorder < MINIMAL\_DOUBLE\_VALUE || bottomBorder > MAXIMUM\_DOUBLE\_VALUE ||

topBorder < MINIMAL\_DOUBLE\_VALUE || topBorder > MAXIMUM\_DOUBLE\_VALUE ||

stepWidth < MINIMAL\_DOUBLE\_VALUE || stepWidth > MAXIMUM\_DOUBLE\_VALUE) {

throw new IntegralValueException(

MINIMAL\_DOUBLE\_VALUE, MAXIMUM\_DOUBLE\_VALUE);

}

\_stepWidth = stepWidth;

\_result = previewResult;

}

public double getTopBorder() { return \_topBorder; }

public double getBottomBorder() { return \_bottomBorder; }

public double getStepWidth() { return \_stepWidth; }

public double getResult() { return \_result; }

public void setResult(double result) { \_result = result; }

}  
  
package calculator.Core.Server;

import java.beans.PropertyChangeListener;

import java.beans.PropertyChangeSupport;

public class CalculationResultAggregator {

private double \_totalResult;

private int \_expectedResponses;

private int \_receivedResponses;

private final PropertyChangeSupport pcs = new PropertyChangeSupport(this);

public synchronized void reset(int expectedResponses) {

\_totalResult = 0.0;

\_receivedResponses = 0;

\_expectedResponses = expectedResponses;

}

public synchronized void addPartialResult(double partial) {

double oldResult = \_totalResult;

\_totalResult += partial;

\_receivedResponses++;

pcs.firePropertyChange("partialResult", oldResult, \_totalResult);

if (\_receivedResponses == \_expectedResponses) {

// Все ответы получены – уведомляем об окончательном результате

pcs.firePropertyChange("finalResult", null, \_totalResult);

}

}

public void addPropertyChangeListener(PropertyChangeListener listener) {

pcs.addPropertyChangeListener(listener);

}

}  
  
package calculator.Core.Server;

import Udp.IntegralRequestPacket;

import Udp.IntegralResponsePacket;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.SocketAddress;

import java.net.SocketException;

import java.util.ArrayList;

import java.util.List;

public class UdpServer {

private final DatagramSocket \_socket;

private final List<SocketAddress> \_clientList = new ArrayList<>();

private final CalculationResultAggregator \_aggregator;

public UdpServer(int port, CalculationResultAggregator aggregator) throws SocketException {

\_socket = new DatagramSocket(port);

\_aggregator = aggregator;

}

// Запускаем прослушивание входящих UDP-сообщений

public void listen() {

new Thread(() -> {

byte[] buf = new byte[1024];

while (true) {

try {

DatagramPacket packet = new DatagramPacket(buf, buf.length);

\_socket.receive(packet);

String msg = new String(packet.getData(), 0, packet.getLength());

// Если сообщение – регистрация, добавляем клиента

if ("REGISTER".equalsIgnoreCase(msg.trim())) {

if (!\_clientList.contains(packet.getSocketAddress())) {

\_clientList.add(packet.getSocketAddress());

System.out.println("The client is registered: " + packet.getSocketAddress());

}

} else {

// Иначе десериализуем ответ с частичным результатом

ByteArrayInputStream bais = new ByteArrayInputStream(packet.getData(), 0, packet.getLength());

ObjectInputStream ois = new ObjectInputStream(bais);

IntegralResponsePacket response = (IntegralResponsePacket) ois.readObject();

System.out.println("Received a response from " + packet.getSocketAddress() +

" - Partial result: " + response.getPartialResult());

\_aggregator.addPartialResult(response.getPartialResult());

}

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}).start();

}

// Отправка запроса конкретному клиенту

public void sendRequest(SocketAddress clientAddress, IntegralRequestPacket request) {

try {

ByteArrayOutputStream baos = new ByteArrayOutputStream();

ObjectOutputStream oos = new ObjectOutputStream(baos);

oos.writeObject(request);

oos.flush();

byte[] data = baos.toByteArray();

DatagramPacket packet = new DatagramPacket(data, data.length, clientAddress);

\_socket.send(packet);

System.out.println("A request has been sent to the client " + clientAddress + ": ["

+ request.getStart() + ", " + request.getEnd() + "]");

} catch (IOException e) {

e.printStackTrace();

}

}

// Разделение общего интервала на части и отправка запросов всем клиентам

public void distributeIntegralCalculation(double bottomBorder, double topBorder, double stepWidth) {

int clientCount = \_clientList.size();

if (clientCount == 0) {

System.out.println("There are no registered clients for calculations!");

return;

}

\_aggregator.reset(clientCount);

double totalInterval = topBorder - bottomBorder;

double subIntervalLength = totalInterval / clientCount;

for (int i = 0; i < clientCount; i++) {

double subStart = bottomBorder + i \* subIntervalLength;

double subEnd = (i == clientCount - 1) ? topBorder : subStart + subIntervalLength;

IntegralRequestPacket request = new IntegralRequestPacket(subStart, subEnd, stepWidth);

sendRequest(\_clientList.get(i), request);

}

}

public List<SocketAddress> getClientList() {

return \_clientList;

}

public void close() {

if (\_socket != null && !\_socket.isClosed()) {

\_socket.close();

}

}

}  
  
package calculator.UI;

import calculator.Core.Data.DataManager;

import calculator.Core.Integral.IntegralValueException;

import calculator.Core.Integral.RecIntegral;

import calculator.Core.Server.CalculationResultAggregator;

import calculator.Core.Server.UdpServer;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

import javax.swing.JTable;

import javax.swing.filechooser.FileNameExtensionFilter;

import javax.swing.table.DefaultTableModel;

public class MainFrame extends javax.swing.JFrame {

// Shadow column

private static final int SHADOW\_COLUMN\_NUMBER = 4;

private static final String SHADOW\_COLUMN\_TITLE = "ShadowColumn";

// Load/Save files

private static final String TEXT\_EXTENSION = ".txt";

private static final String BINARY\_EXTENSION = ".calcbin";

private static final String TEXT\_FILTER\_DESCRIPTION = "Text Files (\*" + TEXT\_EXTENSION + ")";

private static final String BINARY\_FILTER\_DESCRIPTION = "Calc Binary Files (\*" + BINARY\_EXTENSION + ")";

private static final String SAVE\_ERROR\_MESSAGE = "Save error: ";

private static final String LOAD\_ERROR\_MESSAGE = "Load error: ";

private ArrayList<RecIntegral> \_integrals;

private final UdpServer \_udpServer;

private int \_currentCalculationRow = -1;

public MainFrame(UdpServer udpServer, CalculationResultAggregator aggregator) {

initComponents();

\_integrals = new ArrayList<>();

\_udpServer = udpServer;

DataTable.removeColumn(DataTable.getColumn(SHADOW\_COLUMN\_TITLE));

FileSaveMenuAsTextItem.addActionListener(e -> saveAsText());

FileSaveMenuAsBinaryItem.addActionListener(e -> saveAsBinary());

FileLoadMenuFromTextItem.addActionListener(e -> loadFromText());

FileLoadMenuFromBinaryItem.addActionListener(e -> loadFromBinary());

aggregator.addPropertyChangeListener(evt -> {

if ("finalResult".equals(evt.getPropertyName())) {

double finalResult = (double) evt.getNewValue();

javax.swing.SwingUtilities.invokeLater(() -> {

CalculateButton.setEnabled(true);

DefaultTableModel model = (DefaultTableModel) DataTable.getModel();

if (\_currentCalculationRow >= 0) {

model.setValueAt(finalResult, \_currentCalculationRow, 3);

}

});

}

});

}

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

jMenuItem1 = new javax.swing.JMenuItem();

DataTableScrollPanel = new javax.swing.JScrollPane();

DataTable = new javax.swing.JTable();

TopBorderLabel = new javax.swing.JLabel();

BottomBorderLabel = new javax.swing.JLabel();

StepWidthLabel = new javax.swing.JLabel();

TopBorderTextField = new javax.swing.JTextField();

BottomBorderTextField = new javax.swing.JTextField();

StepWidthTextField = new javax.swing.JTextField();

AddButton = new javax.swing.JButton();

DeleteButton = new javax.swing.JButton();

CalculateButton = new javax.swing.JButton();

ClearTableButton = new javax.swing.JButton();

FillTableButton = new javax.swing.JButton();

MenuBar = new javax.swing.JMenuBar();

FileMenu = new javax.swing.JMenu();

FileSaveMenu = new javax.swing.JMenu();

FileSaveMenuAsTextItem = new javax.swing.JMenuItem();

FileSaveMenuAsBinaryItem = new javax.swing.JMenuItem();

FileLoadMenu = new javax.swing.JMenu();

FileLoadMenuFromTextItem = new javax.swing.JMenuItem();

FileLoadMenuFromBinaryItem = new javax.swing.JMenuItem();

jMenuItem1.setText("jMenuItem1");

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

DataTable.setModel(new javax.swing.table.DefaultTableModel(

new Object [][] {

},

new String [] {

"Верхняя граница", "Нижняя граница", "Ширина шага", "Результат", "ShadowColumn"

}

) {

boolean[] canEdit = new boolean [] {

false, false, false, false, false

};

public boolean isCellEditable(int rowIndex, int columnIndex) {

return canEdit [columnIndex];

}

});

DataTableScrollPanel.setViewportView(DataTable);

TopBorderLabel.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

TopBorderLabel.setText("Верхняя граница");

TopBorderLabel.setName(""); // NOI18N

BottomBorderLabel.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

BottomBorderLabel.setText("Ширина шага");

StepWidthLabel.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

StepWidthLabel.setText("Нижняя границы");

TopBorderTextField.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

BottomBorderTextField.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

StepWidthTextField.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

AddButton.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

AddButton.setText("Добавить");

AddButton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseClicked(java.awt.event.MouseEvent evt) {

AddButtonMouseClicked(evt);

}

});

DeleteButton.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

DeleteButton.setText("Удалить");

DeleteButton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseClicked(java.awt.event.MouseEvent evt) {

DeleteButtonMouseClicked(evt);

}

});

CalculateButton.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N

CalculateButton.setText("Вычислить");

CalculateButton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseClicked(java.awt.event.MouseEvent evt) {

CalculateButtonMouseClicked(evt);

}

});

ClearTableButton.setText("Очистить");

ClearTableButton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseClicked(java.awt.event.MouseEvent evt) {

ClearTableButtonMouseClicked(evt);

}

});

FillTableButton.setText("Восстановить");

FillTableButton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseClicked(java.awt.event.MouseEvent evt) {

FillTableButtonMouseClicked(evt);

}

});

FileMenu.setText("File");

FileSaveMenu.setText("Save...");

FileSaveMenuAsTextItem.setText("As txt");

FileSaveMenu.add(FileSaveMenuAsTextItem);

FileSaveMenuAsBinaryItem.setText("As binary");

FileSaveMenu.add(FileSaveMenuAsBinaryItem);

FileMenu.add(FileSaveMenu);

FileLoadMenu.setText("Load...");

FileLoadMenuFromTextItem.setText("From txt");

FileLoadMenu.add(FileLoadMenuFromTextItem);

FileLoadMenuFromBinaryItem.setText("From binary");

FileLoadMenu.add(FileLoadMenuFromBinaryItem);

FileMenu.add(FileLoadMenu);

MenuBar.add(FileMenu);

setJMenuBar(MenuBar);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGap(48, 48, 48)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

.addComponent(TopBorderLabel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(StepWidthLabel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(BottomBorderLabel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(StepWidthTextField, javax.swing.GroupLayout.DEFAULT\_SIZE, 314, Short.MAX\_VALUE)

.addComponent(BottomBorderTextField)))

.addGroup(layout.createSequentialGroup()

.addGap(16, 16, 16)

.addComponent(TopBorderTextField)))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

.addComponent(CalculateButton, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(DeleteButton, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(AddButton, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

.addGap(48, 48, 48))

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(DataTableScrollPanel)

.addContainerGap())

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addContainerGap()

.addComponent(ClearTableButton, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(FillTableButton, javax.swing.GroupLayout.PREFERRED\_SIZE, 317, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGap(15, 15, 15)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(TopBorderLabel)

.addComponent(TopBorderTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(AddButton))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(DeleteButton, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 27, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(StepWidthLabel)

.addComponent(BottomBorderTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(BottomBorderLabel)

.addComponent(StepWidthTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(CalculateButton))

.addGap(18, 18, 18)

.addComponent(DataTableScrollPanel, javax.swing.GroupLayout.PREFERRED\_SIZE, 122, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(ClearTableButton)

.addComponent(FillTableButton))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

pack();

}// </editor-fold>

private void AddButtonMouseClicked(java.awt.event.MouseEvent evt) {

// Check if all fields field

String topBorderFieldText = TopBorderTextField.getText();

String bottomBordeFieldText = BottomBorderTextField.getText();

String stepWidthFieldText = StepWidthTextField.getText();

if (stringIsNullOrEmpty(topBorderFieldText) ||

stringIsNullOrEmpty(bottomBordeFieldText) ||

stringIsNullOrEmpty(stepWidthFieldText)) {

return;

}

// Try to convert

double topBorder;

double bottomBorder;

double stepWidth;

try {

topBorder = Double.parseDouble(topBorderFieldText);

bottomBorder = Double.parseDouble(bottomBordeFieldText);

stepWidth = Double.parseDouble(stepWidthFieldText);

} catch (NumberFormatException e) {

return;

}

// Step must be greater then zero

if (stepWidth <= 0) {

return;

}

// Add data to table

RecIntegral integral;

try {

integral = new RecIntegral(topBorder, bottomBorder, stepWidth);

} catch(IntegralValueException ex){

JOptionPane.showMessageDialog(

this,

ex.getMessage(),

ex.getExceptionName(),

JOptionPane.ERROR\_MESSAGE);

return;

}

\_integrals.add(integral);

AddIntegralToTable(DataTable, integral);

}

private void DeleteButtonMouseClicked(java.awt.event.MouseEvent evt) {

// Check if row selected

int selectedRow = DataTable.getSelectedRow();

if (selectedRow < 0) {

return;

}

// Remove row

DefaultTableModel model = (DefaultTableModel)DataTable.getModel();

\_integrals.remove(

(RecIntegral)model.getValueAt(selectedRow, SHADOW\_COLUMN\_NUMBER));

model.removeRow(selectedRow);

}

private void CalculateButtonMouseClicked(java.awt.event.MouseEvent evt) {

int selectedRow = DataTable.getSelectedRow();

if (selectedRow < 0) {

return;

}

\_currentCalculationRow = selectedRow;

DefaultTableModel model = (DefaultTableModel) DataTable.getModel();

RecIntegral integral = (RecIntegral)model.getValueAt(selectedRow, SHADOW\_COLUMN\_NUMBER);

CalculateButton.setEnabled(false);

\_udpServer.distributeIntegralCalculation(

integral.getBottomBorder(), integral.getTopBorder(), integral.getStepWidth());

System.out.println("A distributed integral calculation is requested.");

}

private void ClearTableButtonMouseClicked(java.awt.event.MouseEvent evt) {

((DefaultTableModel) DataTable.getModel()).setRowCount(0);

}

private void FillTableButtonMouseClicked(java.awt.event.MouseEvent evt) {

DefaultTableModel model = (DefaultTableModel) DataTable.getModel();

model.setRowCount(0);

for (RecIntegral integral : \_integrals) {

AddIntegralToTable(DataTable, integral);

}

}

private void saveAsText() {

JFileChooser chooser = new JFileChooser();

FileNameExtensionFilter textFilter =

new FileNameExtensionFilter(

"Text Files (\*" + TEXT\_EXTENSION + ")",

TEXT\_EXTENSION.replace(".", ""));

chooser.setFileFilter(textFilter);

if (chooser.showSaveDialog(this) == JFileChooser.APPROVE\_OPTION) {

File file = chooser.getSelectedFile();

// Append the .txt extension if it is missing.

if (!file.getName().toLowerCase().endsWith(TEXT\_EXTENSION)) {

file = new File(file.getAbsolutePath() + TEXT\_EXTENSION);

}

try {

DataManager.saveDataAsText(file, \_integrals);

} catch (IOException e) {

JOptionPane.showMessageDialog(this, SAVE\_ERROR\_MESSAGE + e.getMessage());

}

}

}

private void saveAsBinary() {

JFileChooser chooser = new JFileChooser();

FileNameExtensionFilter binFilter = new FileNameExtensionFilter(

BINARY\_FILTER\_DESCRIPTION,

BINARY\_EXTENSION.replace(".", ""));

chooser.setFileFilter(binFilter);

if (chooser.showSaveDialog(this) == JFileChooser.APPROVE\_OPTION) {

File file = chooser.getSelectedFile();

// Append the .calcbin extension if it is missing.

if (!file.getName().toLowerCase().endsWith(BINARY\_EXTENSION)) {

file = new File(file.getAbsolutePath() + BINARY\_EXTENSION);

}

try {

DataManager.saveDataAsBinary(file, \_integrals);

} catch (IOException e) {

JOptionPane.showMessageDialog(this, SAVE\_ERROR\_MESSAGE + e.getMessage());

}

}

}

private void loadFromText() {

JFileChooser chooser = new JFileChooser();

FileNameExtensionFilter textFilter = new FileNameExtensionFilter(

TEXT\_FILTER\_DESCRIPTION,

TEXT\_EXTENSION.replace(".", ""));

chooser.setFileFilter(textFilter);

if (chooser.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION) {

File file = chooser.getSelectedFile();

// Validate the file extension.

if (!file.getName().toLowerCase().endsWith(TEXT\_EXTENSION)) {

JOptionPane.showMessageDialog(this,

"Please select a file with " + TEXT\_EXTENSION + " extension");

return;

}

try {

\_integrals = DataManager.loadDataFromText(file);

for (RecIntegral integral : \_integrals) {

AddIntegralToTable(DataTable, integral);

}

} catch (IOException e) {

JOptionPane.showMessageDialog(this, LOAD\_ERROR\_MESSAGE + e.getMessage());

}

}

}

private void loadFromBinary() {

JFileChooser chooser = new JFileChooser();

FileNameExtensionFilter binFilter = new FileNameExtensionFilter(

BINARY\_FILTER\_DESCRIPTION,

BINARY\_EXTENSION.replace(".", ""));

chooser.setFileFilter(binFilter);

if (chooser.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION) {

File file = chooser.getSelectedFile();

// Validate the file extension.

if (!file.getName().toLowerCase().endsWith(BINARY\_EXTENSION)) {

JOptionPane.showMessageDialog(this,

"Please select a file with " + BINARY\_EXTENSION + " extension");

return;

}

try {

\_integrals = DataManager.loadDataFromBinary(file);

for (RecIntegral integral : \_integrals) {

AddIntegralToTable(DataTable, integral);

}

} catch (IOException e) {

JOptionPane.showMessageDialog(this, LOAD\_ERROR\_MESSAGE + e.getMessage());

} catch (ClassNotFoundException e) {

JOptionPane.showMessageDialog(this, e.getMessage());

}

}

}

private void AddIntegralToTable(JTable table, RecIntegral integral) {

double result = integral.getResult();

((DefaultTableModel)table.getModel()).addRow(

new Object[]{

integral.getTopBorder(),

integral.getBottomBorder(),

integral.getStepWidth(),

result == Double.NaN ? "" : result,

integral});

}

private boolean stringIsNullOrEmpty(String str) {

return str == null || str.trim().isEmpty();

}

// Variables declaration - do not modify

private javax.swing.JButton AddButton;

private javax.swing.JLabel BottomBorderLabel;

private javax.swing.JTextField BottomBorderTextField;

private javax.swing.JButton CalculateButton;

private javax.swing.JButton ClearTableButton;

private javax.swing.JTable DataTable;

private javax.swing.JScrollPane DataTableScrollPanel;

private javax.swing.JButton DeleteButton;

private javax.swing.JMenu FileLoadMenu;

private javax.swing.JMenuItem FileLoadMenuFromBinaryItem;

private javax.swing.JMenuItem FileLoadMenuFromTextItem;

private javax.swing.JMenu FileMenu;

private javax.swing.JMenu FileSaveMenu;

private javax.swing.JMenuItem FileSaveMenuAsBinaryItem;

private javax.swing.JMenuItem FileSaveMenuAsTextItem;

private javax.swing.JButton FillTableButton;

private javax.swing.JMenuBar MenuBar;

private javax.swing.JLabel StepWidthLabel;

private javax.swing.JTextField StepWidthTextField;

private javax.swing.JLabel TopBorderLabel;

private javax.swing.JTextField TopBorderTextField;

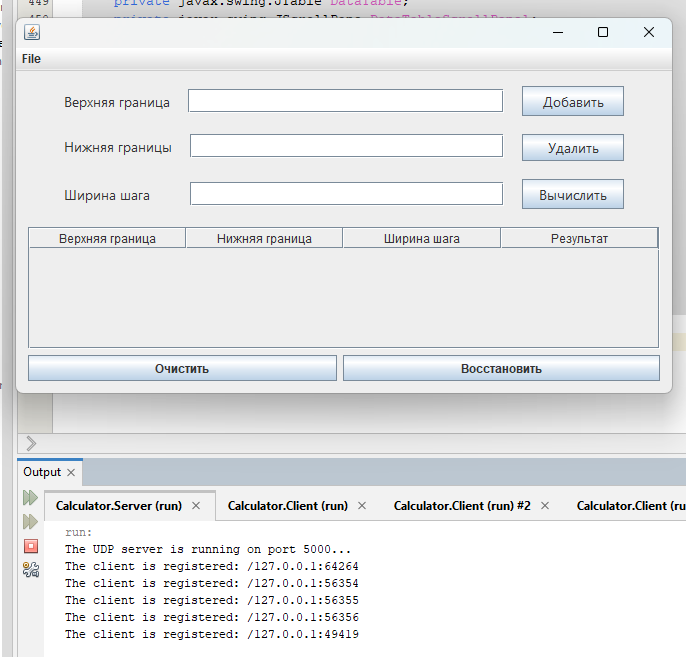
private javax.swing.JMenuItem jMenuItem1;

// End of variables declaration

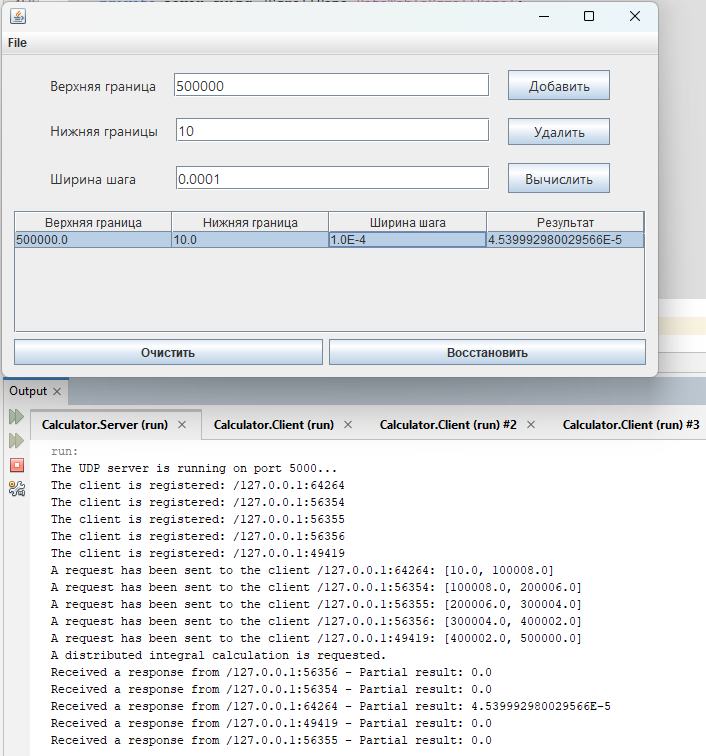
}

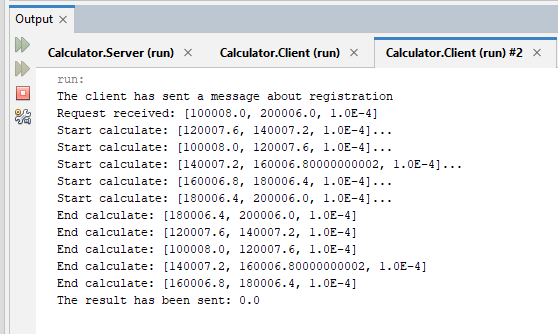
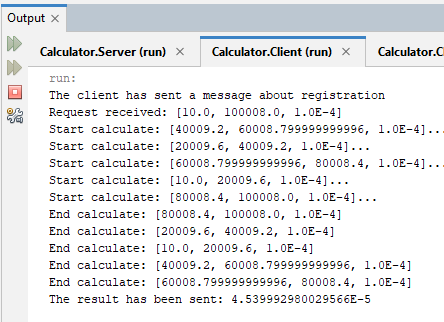
**Результат выполнения программы**

Подключение клиентов

****

Разветвленное вычисление





**Вывод**

Научились создавать клиент-серверные приложения c использованием стандартных классов Java