Министерство науки и высшего образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Программирование на языке Java»

на тему «Многопоточность в Java»

Вариант 5

Выполнили

студенты группы 21ВВП2:

Лукина Е.Ю.

Ванюшин И.А.

Тарасов Н.А.

Приняли:

Юрова О.В.

Карамышева Н.С.

Пенза 2024

**Цель работы**

Научиться создавать многопоточные приложения c использованием стандартных средств языка Java.

**Задание на лабораторную работу**

Модифицировать приложение из предыдущей лабораторной работы, реализовав вычисление определенного интеграла в нескольких дополнительных нитях (число нитей определяется номером варианта), снимая нагрузку с основной нити и предотвращая "подвисание" графического интерфейса. Варианты с номерами до 5 включительно реализуют многопоточность путем наследования от класса Thread, остальные реализуют интерфейс Runnable. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

|  |  |  |
| --- | --- | --- |
| Номер варианта | Функция | Число нитей |
| 5 |  | 5 |

**Ход работы**

Функционально разделили графический интерфейс с вычислением определенного интеграла, выделив под это отдельную нить. Дочерняя нить, в свою очередь, порождает еще 5 дочерних нитей, разделяя изначальную область интегрирования на 5 частей.

Ниже представлен листинг обработки события добавления записи в таблицу.

// "Calculate" case. Calculates the integral with setted parameters.

if (e.getActionCommand().equals("Add"))

{

// Thread here was made to split GUI from calculation stuff

Runnable task = () -> {

int lastI = alRecs.size() - 1; // Saves current row's index to write the result in it later

double result = 0;

if (lastI == -1) return;

try {

double tmpA = ic.A(), tmpB = ic.B(), tmpE = ic.E();

double[] parts = {0, 0, 0, 0, 0};

double integPart = Math.abs(tmpB - tmpA) / 5;

Thread[] thrs = {null, null, null, null, null};

for (int i = 0; i < 5; i++){

int k = i;

// Thread here was made to split Integration on five parts

Runnable part = () -> {

int j = k;

try{

parts[j] = IntegralCalculator.F(tmpA + integPart \* j, tmpA + integPart \* (j + 1), tmpE);

}

catch(IntegralInputException exc){

exc.printStackTrace();

}

};

thrs[i] = new Thread(part);

thrs[i].start();

}

for (int i = 0; i < 5; i++) {

thrs[i].join(); // Syncronization point

result += parts[i]; // Summarizing all parts together

}

((RecIntegral)alRecs.get(lastI)).SetResult(result); // Sets the result to Collection

dtResultsModel.setValueAt(Double.toString(result), lastI, 3); // Sets the result to Table

}

catch(IntegralInputException exc)

{

dtResultsModel.setValueAt("Error", lastI, 3);

JOptionPane.showMessageDialog(

jdFrame,

exc.getMessage(),

"Error",

JOptionPane.ERROR\_MESSAGE

);

}

catch(InterruptedException exc){

exc.printStackTrace();

}

};

Thread CalcThr = new Thread(task);

try

{

if (Thread.activeCount() > 6){

return;

}

ic.SetArea(Double.parseDouble(jtfA.getText()), Double.parseDouble(jtfB.getText()));

ic.SetE(Double.parseDouble(jtfE.getText()));

alRecs.add(new RecIntegral(ic));

jtfE.setText(((RecIntegral)alRecs.getLast()).Dx());

dtResultsModel.addRow(new String[]

{

String.valueOf(ic.A()),

String.valueOf(ic.B()),

String.valueOf(ic.E()),

"Calculating..."

});

CalcThr.start();

}

catch(IntegralInputException exc)

{

JOptionPane.showMessageDialog(

jdFrame,

exc.getMessage(),

"Error",

JOptionPane.ERROR\_MESSAGE

);

}

catch(NumberFormatException exc)

{

JOptionPane.showMessageDialog(

jdFrame,

exc.getMessage(),

"Error",

JOptionPane.ERROR\_MESSAGE

);

}

}

Модифицировали класс RecIntegral, добавив метку Serializable, указывающую, что объекты класса будут сериализованы или десериализованы в процессе исполнения программы.

**Листинг**

IntegralCalculator.java

package lab1;

/\*\*

\*

\* @author User

\*/

public class IntegralCalculator {

public IntegralCalculator()

{

a = 0;

b = 1;

e = 1.0e-4;

}

public IntegralCalculator(double \_a, double \_b, double \_e) throws IntegralInputException

{

if (!(\_a <= 1.0e+6 && \_a >= -1.0e+6)) throw new IntegralInputException("Left limit is out of limit.");

if (!(\_b <= 1.0e+6 && \_b >= -1.0e+6)) throw new IntegralInputException("Right limit is out of limit.");

if (\_e == 0) throw new IntegralInputException("Step cannot be zero.");

if (\_e > Math.abs(\_a) + Math.abs(\_b)) throw new IntegralInputException("Step is greater than integral limit.");

if (\_e <= 1.0e+6 && \_e >= 1.0e-6) e = Math.abs(\_e);

else throw new IntegralInputException("Step is out of limit.");

if (\_a > \_b)

{

a = \_b;

b = \_a;

}

else

{

a = \_a;

b = \_b;

}

}

private double e, a, b;

public double E() { return e; }

public double A() { return a; }

public double B() { return b; }

public void SetE(double \_e) throws IntegralInputException

{

if (\_e == 0) throw new IntegralInputException("Step cannot be zero.");

if (\_e > Math.abs(a) + Math.abs(b)) throw new IntegralInputException("Step is greater than integral limit.");

if (\_e <= 1.0e+6 && \_e >= 1.0e-6) e = Math.abs(\_e);

else throw new IntegralInputException("Step is out of limit.");

}

public void SetArea(double \_a, double \_b) throws IntegralInputException {

if (!(\_a <= 1.0e+6 && \_a >= -1.0e+6)) throw new IntegralInputException("Left limit is out of limit.");

if(!(\_b <= 1.0e+6 && \_b >= -1.0e+6)) throw new IntegralInputException("Right limit is out of limit.");

if (e > Math.abs(\_a) + Math.abs(\_b)) throw new IntegralInputException("Step is greater than integral limit.");

if (\_a > \_b)

{

a = \_b;

b = \_a;

}

else

{

a = \_a;

b = \_b;

}

}

public double F()

{

double result = 0;

double lBound = a, rBound = a + e;

while (rBound < b)

{

result += e \* (Math.exp(-lBound) + Math.exp(-rBound)) / 2;

lBound += e;

rBound += e;

}

if (lBound < b) result += e \* (Math.exp(-lBound) + Math.exp(-b)) / 2;

return result;

}

public static double F(double \_a, double \_b, double \_e) throws IntegralInputException

{

if (!(\_a <= 1.0e+6 && \_a >= -1.0e+6)) throw new IntegralInputException("Left limit is out of limit.");

if (!(\_b <= 1.0e+6 && \_b >= -1.0e+6)) throw new IntegralInputException("Right limit is out of limit.");

if (\_e == 0) throw new IntegralInputException("Step cannot be zero.");

if (\_e > Math.abs(\_a) + Math.abs(\_b)) throw new IntegralInputException("Step is greater than integral limit.");

if (!(\_e <= 1.0e+6 && \_e >= 1.0e-6)) throw new IntegralInputException("Step is out of limit.");

double result = 0;

double lBound = \_a, rBound = \_a + \_e;

while (rBound < \_b)

{

result += \_e \* (Math.exp(-lBound) + Math.exp(-rBound)) / 2;

lBound += \_e;

MyWin.java

package lab1;

import java.awt.Container;

import java.awt.Dimension;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

import java.io.BufferedReader;

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 javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.table.DefaultTableModel;

import javax.swing.JOptionPane;

import javax.swing.JDialog;

import javax.swing.JFileChooser;

public class MyWin extends JFrame implements ActionListener

{

// Some window's elements

private final JTextField jtfA;

private final JTextField jtfB;

private final JTextField jtfE;

private final DefaultTableModel dtResultsModel;

private final JTable jtResults;

private final JDialog jdFrame;

private final JFileChooser jfc;

private ArrayList alRecs;

// Класс для вычисления определенного интеграла

private final IntegralCalculator ic;

// Classes serial number

private static final long serialVersionUID = 1L;

public MyWin()

{

// Initialization block

jdFrame = new JDialog();

alRecs = new ArrayList();

ic = new IntegralCalculator();

jtfA = new JTextField();

jtfB = new JTextField();

jtfE = new JTextField();

jfc = new JFileChooser();

// ErrorDialog configuration

jfc.setDialogTitle("File selection");

jfc.setFileSelectionMode(JFileChooser.FILES\_ONLY);

//----------------==Table Filling==------------------------------

Object[] columnNames = new String[]{"L limit", "R limit", "Step", "Result"};

Object[][] data = new String[][]{};

// Tables initialization

dtResultsModel = new DefaultTableModel(data, columnNames);

dtResultsModel.setColumnIdentifiers(columnNames);

jtResults = new JTable(dtResultsModel);

JScrollPane sp = new JScrollPane(jtResults);

Container c = getContentPane(); // Client rect

c.setLayout(null); // НЕНАВИЖУ КОМПАНОВЩИКИ

//----------------==Left Panel's Filling==-----------------------

// Creates a left sided panel

JPanel jpLeftPanel = new JPanel();

jpLeftPanel.setLayout(null); // АГРЕССИЯ И ЗУБЫ СКРИПЯТ

jpLeftPanel.setBounds(0, 0, 200, 480); // Resizes left panel

// Creates buttons.

// The one that Adds new records & the other that Clears all inputs.

JButton jbtnAdd = new JButton("Add");

JButton jbtnClear = new JButton("Clear");

JButton jbtnDelete = new JButton("Delete");

JButton jbtnLoad = new JButton("Load");

JButton jbtnInFileBin = new JButton("Import file(bin)");

JButton jbtnInFileTxt = new JButton("Import file(txt)");

JButton jbtnOutFileBin = new JButton("Export file(bin)");

JButton jbtnOutFileTxt = new JButton("Export file(txt)");

// Buttons's resize.

jbtnAdd.setBounds(10, 200, 180, 25);

jbtnClear.setBounds(10, 230, 180, 25);

jbtnLoad.setBounds(10, 260, 180, 25);

jbtnDelete.setBounds(10, 290, 180, 25);

jbtnInFileBin.setBounds(10, 320, 180, 25);

jbtnInFileTxt.setBounds(10, 350, 180, 25);

jbtnOutFileBin.setBounds(10, 380, 180, 25);

jbtnOutFileTxt.setBounds(10, 410, 180, 25);

// Adds action listeners for the objects.

jbtnAdd.addActionListener(this);

jbtnClear.addActionListener(this);

jbtnDelete.addActionListener(this);

jbtnLoad.addActionListener(this);

jbtnInFileBin.addActionListener(this);

jbtnInFileTxt.addActionListener(this);

jbtnOutFileBin.addActionListener(this);

jbtnOutFileTxt.addActionListener(this);

// Resizes TextFields

jtfA.setBounds(10, 10, 180, 50);

jtfB.setBounds(10, 70, 180, 50);

jtfE.setBounds(10, 130, 180, 50);

// Sets default values to TextFields

jtfA.setText(Double.toString(ic.A()));

jtfB.setText(Double.toString(ic.B()));

jtfE.setText(Double.toString(ic.E()));

sp.setBounds(200, 0, 427, 480);

// Fills panel with early created buttons & txtFields.

jpLeftPanel.add(jtfA);

jpLeftPanel.add(jtfB);

jpLeftPanel.add(jtfE);

jpLeftPanel.add(jbtnAdd);

jpLeftPanel.add(jbtnClear);

jpLeftPanel.add(jbtnDelete);

jpLeftPanel.add(jbtnLoad);

jpLeftPanel.add(jbtnInFileBin);

jpLeftPanel.add(jbtnInFileTxt);

jpLeftPanel.add(jbtnOutFileBin);

jpLeftPanel.add(jbtnOutFileTxt);

//----------------==Client rect's Filling==-----------------------

// Adds left panel to actually left side of client rect.

c.add(jpLeftPanel);

c.add(sp);

//----------------==Setting window's properties==-----------------

// Window's properties.

setTitle("Integral Calculator"); // Window's title. Obviously.

// Sets preffered size to window.

setPreferredSize(new Dimension(640, 480));

// Exit application on Exit button.

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

pack(); // Setting preffered sizes.

setVisible(true); // Makes window visible.

}

@Override

public void actionPerformed(ActionEvent e) {

// "Calculate" case. Calculates the integral with setted parameters.

if (e.getActionCommand().equals("Add"))

{

// Thread here was made to split GUI from calculation stuff

Runnable task = () -> {

int lastI = alRecs.size() - 1; // Saves current row's index to write the result in it later

double result = 0;

if (lastI == -1) return;

try {

double tmpA = ic.A(), tmpB = ic.B(), tmpE = ic.E();

double[] parts = {0, 0, 0, 0, 0};

double integPart = Math.abs(tmpB - tmpA) / 5;

Thread[] thrs = {null, null, null, null, null};

for (int i = 0; i < 5; i++){

int k = i;

// Thread here was made to split Integration on five parts

Runnable part = () -> {

int j = k;

try{

parts[j] = IntegralCalculator.F(tmpA + integPart \* j, tmpA + integPart \* (j + 1), tmpE);

}

catch(IntegralInputException exc){

exc.printStackTrace();

}

};

thrs[i] = new Thread(part);

thrs[i].start();

}

for (int i = 0; i < 5; i++) {

thrs[i].join(); // Syncronization point

result += parts[i]; // Summarizing all parts together

}

((RecIntegral)alRecs.get(lastI)).SetResult(result); // Sets the result to Collection

dtResultsModel.setValueAt(Double.toString(result), lastI, 3); // Sets the result to Table

}

catch(IntegralInputException exc)

{

dtResultsModel.setValueAt("Error", lastI, 3);

JOptionPane.showMessageDialog(

jdFrame,

exc.getMessage(),

"Error",

JOptionPane.ERROR\_MESSAGE

);

}

catch(InterruptedException exc){

exc.printStackTrace();

}

};

Thread CalcThr = new Thread(task);

try

{

if (Thread.activeCount() > 6){

return;

}

ic.SetArea(Double.parseDouble(jtfA.getText()), Double.parseDouble(jtfB.getText()));

ic.SetE(Double.parseDouble(jtfE.getText()));

alRecs.add(new RecIntegral(ic));

jtfE.setText(((RecIntegral)alRecs.getLast()).Dx());

dtResultsModel.addRow(new String[]

{

String.valueOf(ic.A()),

String.valueOf(ic.B()),

String.valueOf(ic.E()),

"Calculating..."

});

CalcThr.start();

}

catch(IntegralInputException exc)

{

JOptionPane.showMessageDialog(

jdFrame,

exc.getMessage(),

"Error",

JOptionPane.ERROR\_MESSAGE

);

}

catch(NumberFormatException exc)

{

JOptionPane.showMessageDialog(

jdFrame,

exc.getMessage(),

"Error",

JOptionPane.ERROR\_MESSAGE

);

}

}

else if (e.getActionCommand().equals("Clear"))

{

dtResultsModel.setRowCount(0);

}

else if (e.getActionCommand().equals("Load"))

{

dtResultsModel.setRowCount(0);

for (int i = 0; i < alRecs.size(); i++)

{

dtResultsModel.addRow(new String[]

{

((RecIntegral)alRecs.get(i)).LimitL(),

((RecIntegral)alRecs.get(i)).LimitR(),

((RecIntegral)alRecs.get(i)).Dx(),

((RecIntegral)alRecs.get(i)).Result()

});

}

}

else if (e.getActionCommand().equals("Delete"))

{

int i = jtResults.getSelectedRow();

if (i != -1)

{

dtResultsModel.removeRow(i);

alRecs.remove(i);

}

}

else if (e.getActionCommand().equals("Export file(bin)"))

{

ObjectOutputStream out = null;

File fileOpen = null;

if (jfc.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION)

{

fileOpen = jfc.getSelectedFile();

}

else

{

return;

}

try

{

out = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(fileOpen)));

out.writeObject(alRecs);

out.close();

}

catch (IOException ex)

{

ex.printStackTrace();

}

}

else if (e.getActionCommand().equals("Import file(bin)"))

{

ObjectInputStream out = null;

File fileOpen = null;

if (jfc.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION)

{

fileOpen = jfc.getSelectedFile();

}

else

{

return;

}

try

{

out = new ObjectInputStream(new BufferedInputStream(new FileInputStream(fileOpen)));

alRecs = (ArrayList)out.readObject();

out.close();

}

catch (IOException ex)

{

ex.printStackTrace();

}

catch (ClassNotFoundException ex)

{

ex.printStackTrace();

}

}

else if (e.getActionCommand().equals("Export file(txt)"))

{

FileWriter out = null;

File fileOpen = null;

if (jfc.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION)

{

fileOpen = jfc.getSelectedFile();

}

else

{

return;

}

try

{

out = new FileWriter(fileOpen);

for (int i = 0; i < alRecs.size(); i++)

{

RecIntegral tmp = (RecIntegral)alRecs.get(i);

out.write(String.format("%s %s %s %s\n", tmp.LimitL(), tmp.LimitR(), tmp.Dx(), tmp.Result()));

}

out.close();

}

catch (IOException ex)

{

ex.printStackTrace();

}

}

else if (e.getActionCommand().equals("Import file(txt)"))

{

BufferedReader out = null;

String params = "";

String[] splitedPars;

File fileOpen = null;

if (jfc.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION)

{

fileOpen = jfc.getSelectedFile();

}

else

{

return;

}

try

{

out = new BufferedReader(new FileReader(fileOpen));

alRecs.clear();

params = out.readLine();

while (params != null)

{

splitedPars = params.split(" ");

alRecs.add(new RecIntegral(Double.valueOf(splitedPars[0]), Double.valueOf(splitedPars[1]), Double.valueOf(splitedPars[2]), Double.valueOf(splitedPars[3])));

params = out.readLine();

}

out.close();

}

catch (IOException ex)

{

ex.printStackTrace();

}

catch (IntegralInputException ex)

{

ex.printStackTrace();

}

}

else throw new UnsupportedOperationException("Not supported yet."); // Unprocessed action case.

}

// запуск оконного приложения

public static void main(String args[]) {

new MyWin();

}

}

**Результат работы программы**

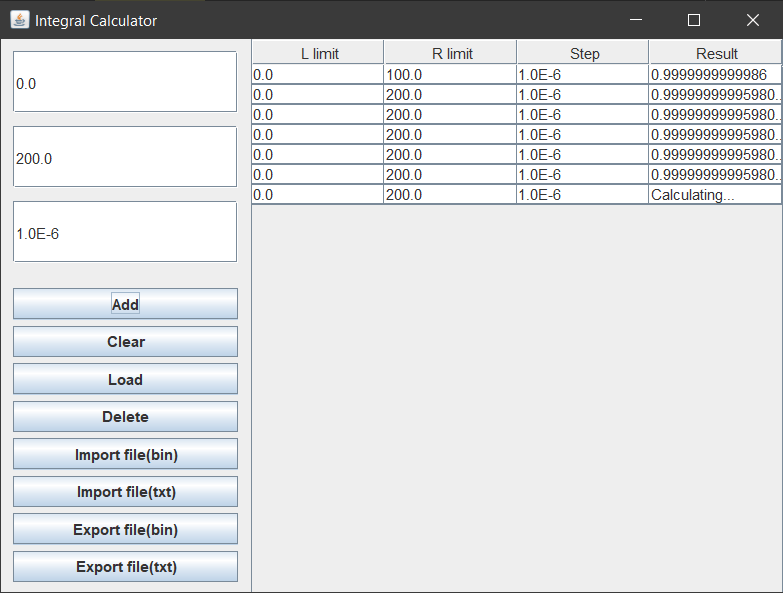
Результаты работы программы представлены на рисунках 1-5. 

Рис.1 – Состояние программы до завершения вычисления

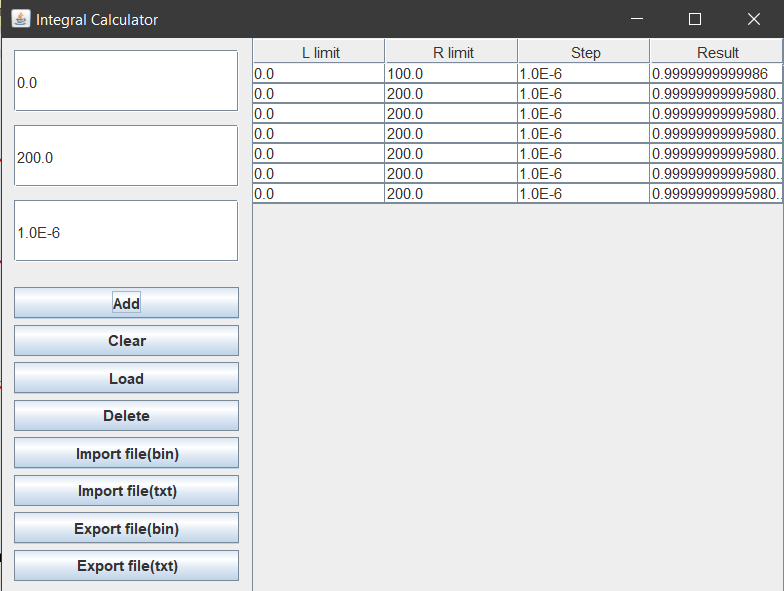


Рис.2 – Состояние программы после завершения вычисления

Результат программы совпал с ожидаемым, следовательно, программа составлена верно.

**Вывод**

В результате выполнения методических указаний к лабораторной работе были получены навыки работы с инструментами для построения многопоточного приложения: функциональным интерфейсом Runnable и классом Thread.