Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Программирование на языке Java»

на тему «Многопоточность в Java»

Выполнили:

студенты группы 20ВВП1

Войнова Д.А.

Зиновьев Я.М.

Проверили:

Юрова О.В.

Карамышева Н.С.

Пенза 2023

**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 потоках (число потоков определяется номером варианта), снимая нагрузку с основного потока и предотвращая "подвисание" графического интерфейса. Варианты с номерами до 5 включительно реализуют многопоточность путем наследования от класса Thread, остальные реализуют интерфейс Runnable.

**Вариант:** 3 - 3 потока

Результат работы программы:

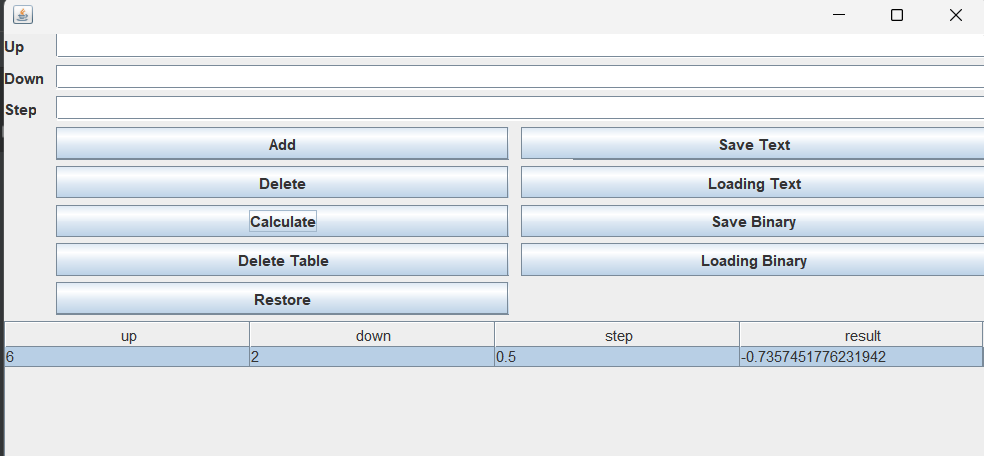


Рис.1

Листинг:

App.java

package main.java;

import javax.swing.\*;

import javax.swing.event.TableModelEvent;

import javax.swing.event.TableModelListener;

import javax.swing.filechooser.FileSystemView;

import javax.swing.table.DefaultTableModel;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.\*;

import java.util.\*;

public class App extends JFrame {

private JTextField up;

private JTextField step;

private JTextField down;

private JButton deleteButton;

private JButton addButton;

private JButton calculateButton;

private JPanel rootPanel;

private JTable table;

private JButton deleteTableButton;

private JButton restoreButton;

private JButton saveTextButton;

private JButton loadingTextButton;

private JButton saveBinaryButton;

private JButton loadingBinaryButton;

private DefaultTableModel model;

private List<RecIntegral> data = new ArrayList<>();

public App() {

setVisible(true);

setSize(800, 600);

setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);

setContentPane(rootPanel);

model = (DefaultTableModel) table.getModel();

model.addColumn("up");

model.addColumn("down");

model.addColumn("step");

model.addColumn("result");

addButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String up = App.this.up.getText();

String down = App.this.down.getText();

String step = App.this.step.getText();

try {

if ((Float.*parseFloat*(up) < 0.000001) || (Float.*parseFloat*(up) > 1000000) || (Float.*parseFloat*(

down) < 0.000001) || (Float.*parseFloat*(down) > 1000000) || (Float.*parseFloat*(

step) < 0.000001) || (Float.*parseFloat*(step) > 1000000)) {

throw new ExceptionInput("wrong input");

}

} catch (ExceptionInput ex) {

App.this.up.setText("");

App.this.down.setText("");

App.this.step.setText("");

new ExceptionWarning();

return;

}

try {

if (Float.*parseFloat*(up) < Float.*parseFloat*(down)) {

throw new ExceptionInput("wrong input");

}

} catch (ExceptionInput ex) {

App.this.up.setText("");

App.this.down.setText("");

App.this.step.setText("");

new ExceptionUpDown();

return;

}

model.addRow(new String[]{up, down, step, "0"});

App.this.up.setText("");

App.this.down.setText("");

App.this.step.setText("");

data.add(new RecIntegral(Arrays.*stream*(new String[]{up, down, step, "0"}).toList()));

}

});

deleteButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int temp = table.getSelectedRow();

if (temp != -1) {

model.removeRow(temp);

data.remove(temp);

}

}

});

calculateButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int temp = table.getSelectedRow();

if (temp == -1) {

return;

}

Vector localData = model.getDataVector().get(temp);

float max = Float.*parseFloat*((String) localData.get(0));

float min = Float.*parseFloat*((String) localData.get(1));

float step = Float.*parseFloat*((String) localData.get(2));

int lenModStep = (int) ((int) (max - Math.*abs*(min)) % step);

int stepsCount = (int) ((int) (max - Math.*abs*(min)) / step);

int threadCount = (lenModStep == 0 ? Math.*min*(stepsCount, 3) : stepsCount + 1 > 3 ? 3 : stepsCount);

ResultResource rs = new ResultResource(model, data, temp);

model.setValueAt(0,temp, 3);

data.get(temp).setDataByIndex(3, String.*valueOf*(0));

List<CalculationThread> threads = new ArrayList<>();

if (threadCount == 3) {

int steps = stepsCount / 3;

threads.add(new CalculationThread(min, min + steps \* step, step, rs));

threads.add(

new CalculationThread(min + steps \* step, min + steps \* 2 \* step, step,

rs));

threads.add(new CalculationThread(min + steps \* 2 \* step, max, step, rs));

} else if (threadCount == 2) {

int steps = stepsCount / 2;

threads.add(new CalculationThread(min, min + steps \* step, step, rs));

threads.add(new CalculationThread(min + steps \* step, max, step, rs));

} else {

threads.add(new CalculationThread(min, max, step, rs));

}

threads.forEach(CalculationThread::start);

}

});

restoreButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int temp = table.getRowCount();

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

model.removeRow(0);

}

for (RecIntegral element : data) {

model.addRow(element.getRecord().toArray());

}

}

});

deleteTableButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int temp = table.getRowCount();

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

model.removeRow(0);

}

}

});

model.addTableModelListener(new TableModelListener() {

@Override

public void tableChanged(TableModelEvent e) {

if (e.getType() == TableModelEvent.UPDATE) {

data.get(table.getSelectedRow()).setDataByIndex(table.getSelectedColumn(),

(String) model.getDataVector().get(table.getSelectedRow()).get(table.getSelectedColumn()));

}

}

});

saveBinaryButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

ObjectOutputStream out = null;

int temp = table.getSelectedRow();

if (temp == -1) {

return;

}

try {

out = new ObjectOutputStream(new BufferedOutputStream(

new FileOutputStream("BinaryStringNumber" + temp + ".txt")));

out.writeObject(data.get(temp));

out.close();

} catch (IOException ignored) {

}

}

});

loadingBinaryButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser(FileSystemView.getFileSystemView().getHomeDirectory());

fileChooser.showOpenDialog(null);

ObjectInputStream in = null;

RecIntegral restObj = null;

int temp = table.getRowCount();

if (temp != -1) {

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

model.removeRow(0);

}

}

try {

in = new ObjectInputStream(new BufferedInputStream(

new FileInputStream(fileChooser.getSelectedFile().getAbsolutePath())));

restObj = (RecIntegral) in.readObject();

data.add(restObj);

model.addRow(restObj.getRecord().toArray());

} catch (IOException | ClassNotFoundException ex) {

ex.printStackTrace();

}

}

});

saveTextButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

ObjectOutputStream out = null;

int temp = table.getSelectedRow();

if (temp == -1) {

return;

}

try {

out = new ObjectOutputStream(new BufferedOutputStream(

new FileOutputStream("TextStringNumber" + temp + ".txt")));

out.writeObject(data.get(temp).toString());

out.close();

} catch (IOException ignored) {

}

}

});

loadingTextButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser(FileSystemView.getFileSystemView().getDefaultDirectory());

fileChooser.showOpenDialog(null);

ObjectInputStream in = null;

RecIntegral restObj = null;

int temp = table.getRowCount();

if (temp != -1) {

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

model.removeRow(0);

}

}

try {

in = new ObjectInputStream(new BufferedInputStream(

new FileInputStream(fileChooser.getSelectedFile().getAbsolutePath())));

restObj = RecIntegral.fromString((String) in.readObject());

data.add(restObj);

model.addRow(restObj.getRecord().toArray());

} catch (IOException | ClassNotFoundException ex) {

ex.printStackTrace();

}

}

});

}

public static void main(String[] args) {

new App();

}

private void createUIComponents() {

table = new JTable() {

@Override

public boolean isCellEditable(int row, int column) {

return column != 3;

}

};

}

}

CalculationThread.java

package main.java;

public class CalculationThread extends Thread {

private double min;

private double max;

private double step;

private final ResultResource result;

public CalculationThread(double min, double max, double step, ResultResource result) {

this.min = min;

this.max = max;

this.step = step;

this.result = result;

}

public void run() {

double localResult = 0;

for (double i = min; i < max - step; i += step) {

if (i > max) {

localResult += (Math.*cos*(i - step) + Math.*cos*(max)) / 2 \* step;

} else {

localResult += (Math.*cos*(i) + Math.*cos*(i + step)) / 2 \* step;

}

}

synchronized (result) {

result.getModel().setValueAt(

Double.*parseDouble*(String.*valueOf*(result.getModel().getDataVector().get(result.getSelectedRow()).get(3))) + localResult,

result.getSelectedRow(), 3);

result.getData().get(result.getSelectedRow()).setDataByIndex(3,

result.getData().get(result.getSelectedRow()).getRecord().get(3) + localResult);

}

}

}

Вывод: в ходе работы мы научились создавать многопоточные приложения c использованием стандартных средств языка Javа.