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 «Многопоточная работа в Java»

Выполнили:

студент группы 20ВВП1

Беляков А. М.

Проверили:

Юрова О.В.

Карамышева Н.С.

Пенза 2023

**Цель работы:** научится создавать многопоточные приложения с использованием стандартных средств языка java.

**Задание на лабораторную работу:**

Модифицировать приложение из предыдущей лабораторной работы, реализовав много поточную обработку интегралла.

Листинг

App.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.ArrayList;  
import java.util.List;  
import java.util.Vector;  
  
public class Application extends JFrame {  
 private static final List<String> *tableHeader* = List.*of*(new String[]{"step", "min", "max", "result"});  
 private static final List<List<String>> *startData* = List.*of*(List.*of*(new String[]{"0.01", "1", "5"}),  
 List.*of*(new String[]{"0.001", "-2", "5"}));  
 private static final int *NON\_EDITABLE\_COLUMN* = 3;  
 private JTextField stepTextField;  
 private JTextField minTextField;  
 private JTextField maxTextField;  
 private JButton addButton;  
 private JButton deleteButton;  
 private JButton calculateButton;  
 private JTable table;  
 private JPanel rootPanel;  
 private JButton deleteTableButton;  
 private JButton uploadButton;  
 private JButton saveBinaryButton;  
 private JButton saveTextButton;  
 private JButton loadingBinaryButton;  
 private JButton loadingTextButton;  
 private DefaultTableModel defaultTableModel;  
 private List<RecIntegral> data;  
  
 public Application() {  
 super("Lab\_1");  
 setContentPane(rootPanel);  
 setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 setVisible(true);  
 setSize(800, 600);  
 addButton.addActionListener(new AddButtonActionListener());  
 deleteButton.addActionListener(new DeleteButtonActionListener());  
 calculateButton.addActionListener(new CalculateButtonActionListener());  
 defaultTableModel.addTableModelListener(new ChangeTableListener());  
 deleteTableButton.addActionListener(new DeleteTableListener());  
 uploadButton.addActionListener(new UploadTableListener());  
  
 saveBinaryButton.addActionListener(new SaveBinaryButtonActionListener());  
 loadingBinaryButton.addActionListener(new LoadingBinaryButtonActionListener());  
 saveTextButton.addActionListener(new SaveTextActionListener());  
 loadingTextButton.addActionListener(new LoadingTextButtonActionListener());  
 }  
  
 public static void main(String[] args) {  
 new Application();  
 }  
  
 private void createUIComponents() {  
 data = new ArrayList<>();  
 table = new JTable() {  
 @Override  
 public boolean isCellEditable(int row, int column) {  
 return column != *NON\_EDITABLE\_COLUMN*;  
 }  
 };  
 defaultTableModel = (DefaultTableModel) table.getModel();  
 *tableHeader*.forEach(defaultTableModel::addColumn);  
 *startData*.forEach(this::addRow);  
  
 }  
  
 private void addRow(List<String> data) {  
 defaultTableModel.addRow(data.toArray());  
 this.data.add(new RecIntegral(data));  
 }  
  
 private boolean inRange(double arg) {  
 return arg > 0.000001 && arg < 1000000;  
 }  
  
 private class SaveBinaryButtonActionListener implements 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) {  
 }  
 }  
 }  
  
 private class LoadingBinaryButtonActionListener implements 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++) {  
 defaultTableModel.removeRow(0);  
 }  
 }  
 try {  
 in = new ObjectInputStream(new BufferedInputStream(  
 new FileInputStream(fileChooser.getSelectedFile()  
 .getAbsolutePath())));  
 restObj = (RecIntegral) in.readObject();  
 data.add(restObj);  
 defaultTableModel.addRow((Vector<?>) restObj.getData());  
 } catch (IOException | ClassNotFoundException ex) {  
 ex.printStackTrace();  
 }  
 }  
 }  
  
 private class SaveTextActionListener implements 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) {  
 }  
 }  
 }  
  
 private class LoadingTextButtonActionListener implements 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++) {  
 defaultTableModel.removeRow(0);  
 }  
 }  
 try {  
 in = new ObjectInputStream(new BufferedInputStream(  
 new FileInputStream(fileChooser.getSelectedFile()  
 .getAbsolutePath())));  
 restObj = RecIntegral.*fromString*((String) in.readObject());  
 data.add(restObj);  
 defaultTableModel.addRow((Vector<?>) restObj.getData());  
  
 } catch (IOException | ClassNotFoundException ex) {  
 ex.printStackTrace();  
 }  
 }  
 }  
  
 private class AddButtonActionListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 List<String> row = new ArrayList<>();  
 try {  
 if ((!stepTextField.getText()  
 .isEmpty() && inRange(Double.*parseDouble*(stepTextField.getText()))) &&  
 (!minTextField.getText()  
 .isEmpty() && inRange(Double.*parseDouble*(minTextField.getText()))) &&  
 (!maxTextField.getText()  
 .isEmpty() && inRange(Double.*parseDouble*(maxTextField.getText())))) {  
 row.add(stepTextField.getText());  
 stepTextField.setText("");  
 row.add(minTextField.getText());  
 minTextField.setText("");  
 row.add(maxTextField.getText());  
 maxTextField.setText("");  
 addRow(row);  
 } else {  
 stepTextField.setText("");  
 minTextField.setText("");  
 maxTextField.setText("");  
 throw new WrongInputException();  
 }  
 } catch (WrongInputException exc) {  
 new WrongInputDialog();  
 }  
 }  
 }  
  
 private class DeleteButtonActionListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedRow = table.getSelectedRow();  
 if (selectedRow != -1) {  
 defaultTableModel.removeRow(selectedRow);  
 data.remove(selectedRow);  
 }  
 }  
 }  
  
 private class CalculateButtonActionListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedRow = table.getSelectedRow();  
 if (selectedRow == -1) {  
 return;  
 }  
 Vector args = defaultTableModel.getDataVector()  
 .get(selectedRow);  
 double step = Double.*parseDouble*((String) args.get(0));  
 double min = Double.*parseDouble*((String) args.get(1));  
 double max = Double.*parseDouble*((String) args.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(defaultTableModel, data, selectedRow);  
 defaultTableModel.setValueAt(0,selectedRow, 3);  
 data.get(selectedRow).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);  
 }  
 }  
  
 private class ChangeTableListener implements TableModelListener {  
 public void tableChanged(TableModelEvent e) {  
 if (e.getType() == TableModelEvent.*UPDATE*) {  
 data.get(table.getSelectedRow())  
 .set(table.getSelectedColumn(),  
 (String) defaultTableModel.getDataVector()  
 .get(table.getSelectedRow())  
 .get(table.getSelectedColumn()));  
 }  
 }  
 }  
  
 private class DeleteTableListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int items = defaultTableModel.getRowCount();  
 for (int i = 0; i < items; i++) {  
 defaultTableModel.removeRow(0);  
 }  
 }  
 }  
  
 private class UploadTableListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int items = defaultTableModel.getRowCount();  
 for (int i = 0; i < items; i++) {  
 defaultTableModel.removeRow(0);  
 }  
 data.forEach(i -> defaultTableModel.addRow(i.getData()  
 .toArray()));  
 }  
 }  
}

CalculationThread.java

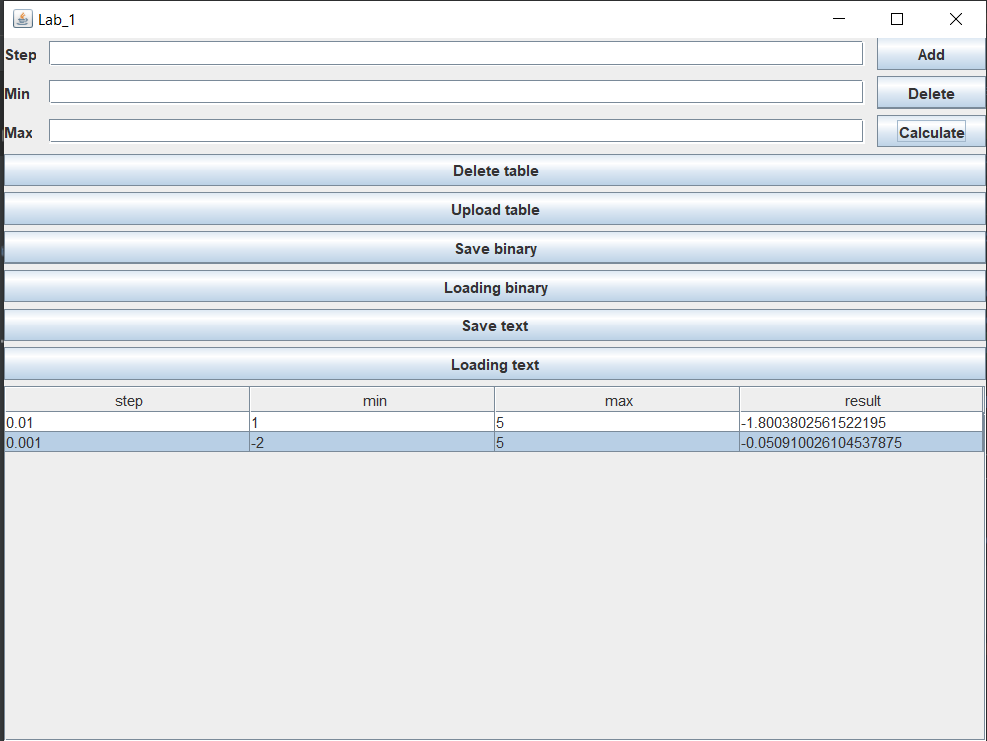
public class CalculationThread extends Thread {  
 private double min;  
 private double max;  
 private double step;  
 private final ResultResource result;  
  
 public CalculationThread(double min, double max, double step, ResultResource result) {  
 this.min = min;  
 this.max = max;  
 this.step = 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()).getData().get(3) + localResult);  
 }  
  
 }  
}

RecIntegral.java

import java.util.ArrayList;  
import java.util.List;  
  
public class RecIntegral {  
  
 private List<String> data;  
  
 public RecIntegral(List<String> data) {  
 this.data = new ArrayList<>(data);  
 if(this.data.size()==3){this.data.add("0");}  
 }  
  
 public void set(int index, String data) {  
 this.data.set(index,data);  
 }  
  
 public List<String> getData() {  
 return data;  
 }  
  
 public static RecIntegral fromString(String inputData) {  
 StringBuffer sb =new StringBuffer(inputData);  
 sb.delete(0,20);  
 sb.delete(sb.length()-2,sb.length());  
 List<String> localstring = List.*of*(sb.toString().split(", "));  
 return new RecIntegral(localstring);  
 }  
  
 public void setDataByIndex(int index, String record) {  
 this.data.set(index, record);  
 }  
 @Override  
 public String toString() {  
 return "RecIntegral{" +  
 "data=" + data +  
 '}';  
 }  
}

ResultResourse.java

import javax.swing.table.DefaultTableModel;  
import java.util.ArrayList;  
import java.util.List;  
  
public class ResultResource {  
 int selectedRow;  
 private DefaultTableModel model;  
 private List<RecIntegral> data = new ArrayList<>();  
  
 public ResultResource(DefaultTableModel model, List<RecIntegral> data , int selectedRow) {  
 this.model = model;  
 this.data = data;  
 this.selectedRow = selectedRow;  
 }  
  
 public int getSelectedRow() {  
 return selectedRow;  
 }  
  
 public void setSelectedRow(int selectedRow) {  
 this.selectedRow = selectedRow;  
 }  
  
 public DefaultTableModel getModel() {  
 return model;  
 }  
  
 public void setModel(DefaultTableModel model) {  
 this.model = model;  
 }  
  
 public List<RecIntegral> getData() {  
 return data;  
 }  
  
 public void setData(List<RecIntegral> data) {  
 this.data = data;  
 }  
}



Вывод: мы научились создавать многопоточные приложения.