МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Севастопольский государственный университет»

Институт информационных технологий и управления в технических системах

кафедра Информационные системы

09.03.02 Информационные системы и технологии (уровень бакалавриата)

Лабораторная работа №4

по дисциплине: «Платформа Java»

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ РАЗРАБОТКИ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА В JAVA ПРИЛОЖЕНИЯХ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ SWING

Вариант – 8

Выполнил

студент 3 курса группы ИС/б-33-о

Генералов Николай Николаевич

Отметка о зачете\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(дата)

Проверил

ст. пр. \_\_\_\_\_\_\_\_\_\_\_ Кузнецов С. А.

(должность) (подпись) (фамилия, инициалы)

г. Севастополь

2018 г.

# ЦЕЛЬ РАБОТЫ

В ходе выполнения данной лабораторной работы необходимо ознакомиться с особенностями инструментария библиотеки SWING для создания графического интерфейса приложений на языке Java и приобрести практические навыки создания Java-программ с графическим интерфейсом, позволяющим пользователю осуществлять взаимодействие с программой: задавать исходные данные, просматривать результаты работы программы в удобном виде.

# ПОСТАНОВКА ЗАДАЧИ

Создать Java приложение с графическим интерфейсом пользователя, реализующее добавление, редактирование, сортировку и удаление данных заданного по варианту типа информации Auto. Данные отображать в виде таблицы. Реализовать поля ввода для добавления и редактирования новых записей. Предусмотреть возможность загрузки информации из текстового файла и сохранения в текстовый файл. При написании программы следует учесть следующие требования и рекомендации:

1. Создать публичный класс, представляющий Автомобиль(Марка, Год выпуска, Объем двигателя, Максимальная скорость), т.е. строку таблицы.
2. Создать модель данных таблицы. Для этого создать класс, расширяющий абстрактный класс AbstractTableModel. Создать в нем объект коллекции типа Auto.

Переопределить методы:

* public Class getColumnClass(int columnIndex)
* public int getColumnCount()
* public String getColumnName(int columnIndex)
* public Object getValueAt(int rowIndex, int columnIndex)
* public boolean isCellEditable(int rowIndex, int columnIndex)
* public void setValueAt(Object value, int rowIndex, int columnIndex)

Определить методы:

* public void addRow() – добавления элемента (строки)
* public void deleteRow(String Поле\_1) – удаления элемента по значению поля 1
* public void updateRow(int row, ) – изменения элемента заданной строки)

1. Для реализации окна приложения реализовать дочерний класс JFrame.
2. Для представления таблицы с данными использовать компонент класса JTable, разместив его в контейнере JScrollPane (для возможности добавления полос прокрутки).
3. Поля ввода для добавления и редактирования данных реализовать текстовыми компонентами JTextField. Каждое поле снабдить подписью при помощи компонентов JLabel.
4. Для выполнения действий открытия файла, добавления, изменения, удаления записи, сортировки и сохранения файла реализовать соответствующие кнопки, с использованием компонентов JButton и добавлением ActionListener. Реализовать загрузку записи в поля для редактирования при щелчке по строке таблицы.
5. Для выбора файла при открытии и сохранении использовать компонент JFileChooser.
6. Удаление и сортировка элементов должно проходить по ключевому полю «Максимальная скорость». Направление сортировки – возрастание.

# ХОД РАБОТЫ

При запуске разработанного приложения открывается основное окно программы, которое содержит рабочую область, где будет располагаться таблица и набор кнопок для манипуляции данными в этой таблице.

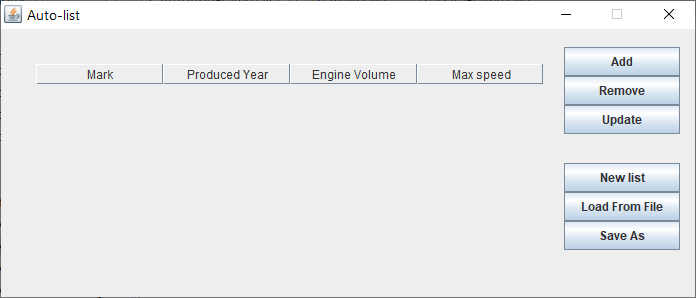


Рисунок 1 – Стартовое окно программы

Для начала работы возможно несколько сценариев действий.

Одним из сценариев является заполнение таблицы в ручную. Нажав на кнопку Add, будет открыто дополнительное окно, в котором располагаются поля для ввода информации, а так же 2 кнопки для добавления заполненных полей в таблицу или их очистки.

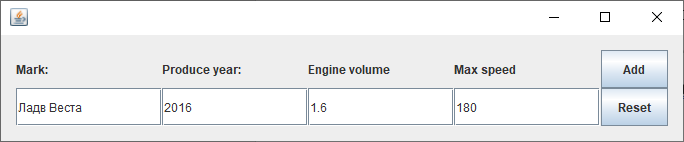


Рисунок 2 – Окно добавления данных в таблицу

При нажатии кнопки «Add», в случае корректного ввода данных, данные добавятся в таблицу и она обновится, в противном случае будет показано диалоговое окно с ошибкой.

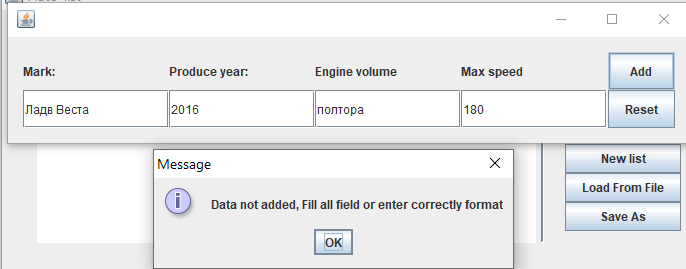


Рисунок 3 – Результат обработки ошибки некорректного ввода данных

При обновлении данных, происходит их сортировка и рендеринг в окно программы.

Вторым возможным сценарием является извлечение данных из файла. Для этого необходим нажать кнопку “Load From File” и выбрать текстовый файл, который имеет структуру построчного перечисления полей, при этом данные могут быть набраны в текстовом редакторе или сохранены с помощью программы по нажатию кнопки “Save As”.

Для манипуляции существующими данными предусмотрены функции удаления данных и их обновления.

Операции удаления предусматривает как одиночное, так и групповое удаление элементов таблицы.

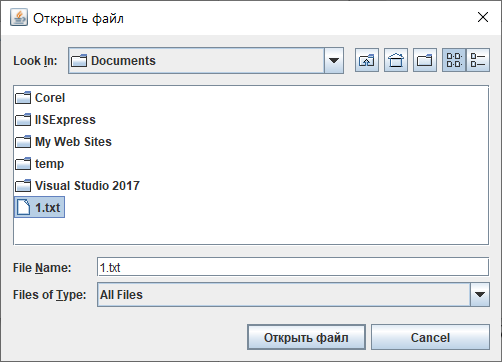


Рисунок 4 – Диалоговое окно выбора файла.

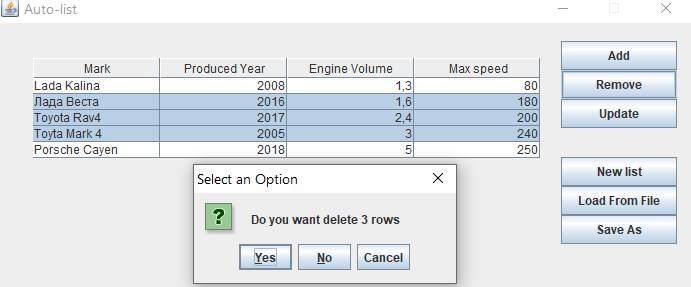


Рисунок 5 – Диалоговое окно подтверждения операции удаления данных

При обновлении данных открывается такое же диалоговое окно, как и при добавлении новых записей, но уже заполненное данными выбранной ячейки таблицы

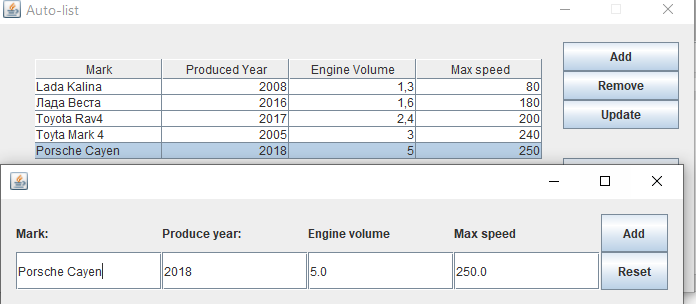


Рисунок 6 – Окно редактирования данных таблицы

Если перед нажатием кнопки «Remove» или «Update» не было выбрано ни одной строки таблицы, будет выведено соответствующее диалоговое окно с ошибкой.

# ВЫВОД

В ходе выполнения лабораторной работы были получены теоретические знания для разработки приложений с графическим интерфейсом, с использованием инструментария библиотеки Java Swing. Полученные знания были закреплены на практике при создании приложения, которое позволяет манипулировать данными, имеющими структуру связного списка, с помощью таблицы.

Данное приложение позволяет производить добавление, удаление и редактирование данных в таблице, а так же сохранение и загрузку данных из файла.

По результатам работы программы можно сделать вывод, что поставленная цель достигнута.

Приложение А

Текст программы

“Auto.java”

import java.util.Scanner;

public class Auto implements Comparable<Auto>{

private String Mark;

private int ProductionYear;

private double EngineVolume;

private double MaxSpeed;

final private static int DATA\_FIELDS\_COUNT = 4;

public Auto(String mark, int productionYear, double engineVolume, double maxSpeed) {

setMark(mark);

setProductionYear(productionYear);

setEngineVolume(engineVolume);

setMaxSpeed(maxSpeed);

}

public Auto(String mark, String productionYear, String engineVolume, String maxSpeed) {

setMark(mark);

setProductionYear(Integer.parseInt(productionYear));

setEngineVolume(Double.parseDouble(engineVolume));

setMaxSpeed(Double.parseDouble(maxSpeed));

}

public static Auto ReadFromScanner(Scanner scanner) {

String tempMark = null;

int prodYear = 0;

double engineVol = 0.0;

double maxSpeed = 0.0;

if (scanner.hasNextLine()) {

tempMark = scanner.nextLine();

}

if (scanner.hasNextInt()) {

prodYear = scanner.nextInt();

}

if (scanner.hasNextDouble()) {

engineVol = scanner.nextDouble();

}

if (scanner.hasNextDouble()) {

maxSpeed = scanner.nextDouble();

}

if (scanner.hasNextLine()) scanner.nextLine();

return new Auto(tempMark, prodYear, engineVol, maxSpeed);

}

//Sort MaxSpeed Asc LinkedList

public String getMark() {

return Mark;

}

public void setMark(String mark) {

Mark = mark;

}

public int getProductionYear() {

return ProductionYear;

}

public void setProductionYear(int productionYear) {

ProductionYear = productionYear;

}

public double getEngineVolume() {

return EngineVolume;

}

public void setEngineVolume(double engineVolume) {

EngineVolume = engineVolume;

}

public double getMaxSpeed() {

return MaxSpeed;

}

public void setMaxSpeed(double maxSpeed) {

MaxSpeed = maxSpeed;

}

@Override

public int compareTo(Auto auto) {

// TODO Auto-generated method stub

return Double.compare(this.getMaxSpeed(), auto.getMaxSpeed());

}

static public int getDataFieldsCount() {

return DATA\_FIELDS\_COUNT;

}

static public String getFieldName(int fieldNum) {

switch (fieldNum) {

case 0:{

return "Mark";

}

case 1:{

return "Produced Year";

}

case 2:{

return "Engine Volume";

}

case 3:{

return "Max speed";

}

}

return null;

}

public Object getValueAtField(int fieldNum) {

switch (fieldNum) {

case 0:{

return this.getMark();

}

case 1:{

return this.getProductionYear();

}

case 2:{

return this.getEngineVolume();

}

case 3:{

return this.getMaxSpeed();

}

}

return null;

}

public void setValueAtField(Object value, int filedNum) {

switch (filedNum) {

case 0:{

this.setMark((String)value);

return;

}

case 1:{

this.setProductionYear((int)value);

return;

}

case 2:{

this.setEngineVolume((double)value);

return;

}

case 3:{

this.setMaxSpeed((double)value);

return;

}

}

}

}

“AutoCollection.java”

import java.io.File;

import java.io.FileNotFoundException;

import java.io.PrintWriter;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.Locale;

import java.util.Scanner;

@SuppressWarnings("serial")

public class AutoCollection extends LinkedList<Auto>{

public AutoCollection() {

super();

}

public AutoCollection(LinkedList<Auto> data) {

super(data);

}

public Iterator<Auto> find(String markOfAuto) {

Iterator<Auto> iter = this.iterator();

while(iter.hasNext()) {

if (iter.next().getMark().equals(markOfAuto)) {

return iter;

}

}

return null;

}

public void LoadFromFile(File file) {

File fin = null;

Scanner scanner = null;

try {

fin = file;

scanner = new Scanner(fin);

scanner.useLocale(Locale.ENGLISH);

while(scanner.hasNextLine()) {

Auto tempAutoData = Auto.ReadFromScanner(scanner);

this.add(tempAutoData);

}

} catch(FileNotFoundException exception) {

System.out.println("exception");

} catch(Exception exception) {

System.out.println("exception");

}

finally {

if (scanner != null) {

scanner.close();

}

}

}

public void SaveToFile(File file) {

PrintWriter fout = null;

try {

fout = new PrintWriter(file);

Iterator<Auto> autoIter = this.iterator();

while(autoIter.hasNext()) {

Auto auto = autoIter.next();

fout.println(auto.getMark());

fout.println(auto.getProductionYear());

fout.println(auto.getEngineVolume());

fout.println(auto.getMaxSpeed());

}

} catch(FileNotFoundException exception) {

System.out.println("exception");

} catch(Exception exception) {

System.out.println("exception");

}

finally {

if (fout != null) {

fout.close();

}

}

}

}

“AutoAppFrame.java”

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.border.EmptyBorder;

public class AutoAppFrame extends JFrame {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private JPanel contentPane;

private AutoTableModel tableModel;

private JTable tableData;

/\*\*

\* Create the frame.

\*/

public AutoAppFrame() {

super("Auto-list");

// super("JavaPlatform\_lab\_4\_Auto-list");

try {

this.BuildMainFrame();

this.BuildTableField();

this.BuildButtons();

} catch(Exception ex) {

JOptionPane.showMessageDialog(AutoAppFrame.this, ex);

}

}

public static int GetRelativeWindowWidth(JFrame element, int proc) {

return (int)element.getWidth() \* proc / 100;

}

public static int GetRelativeWindowHeight(JFrame element, int proc) {

return (int)element.getHeight() \* proc / 100;

}

private void BuildMainFrame() {

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setBounds(100, 100, 700, 300);

this.setResizable(false);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(15, 15, 15, 15));

contentPane.setLayout(new BorderLayout(1,2));

this.setContentPane(contentPane);

}

private void BuildButtons() {

JPanel btnPanel = new JPanel();

btnPanel.setLayout(new GridLayout(8,1));

JButton addBtn = new JButton("Add");

JButton removeBtn = new JButton("Remove");

JButton updateBtn = new JButton("Update");

JButton hiddenBtn = new JButton();

JButton newListBtn = new JButton("New list");

JButton loadBtn = new JButton("Load From File");

JButton saveBtn = new JButton("Save As");

btnPanel.add(addBtn);

btnPanel.add(removeBtn);

btnPanel.add(updateBtn);

btnPanel.add(hiddenBtn);

btnPanel.add(newListBtn);

btnPanel.add(loadBtn);

btnPanel.add(saveBtn);

hiddenBtn.setVisible(false);

contentPane.add(btnPanel,BorderLayout.EAST);

addBtn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new DataFrame(tableModel);

}

});

removeBtn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int rowCount = tableData.getSelectedRowCount();

if (rowCount > 1) {

int result = JOptionPane.showConfirmDialog(AutoAppFrame.this, "Do you want delete " + rowCount + " rows");

if (result == JOptionPane.OK\_OPTION) {

tableModel.removeRange(tableData.getSelectedRows());

tableModel.fireTableDataChanged();

}

} else if (rowCount == 1){

int selectedRow = tableData.getSelectedRow();

int result = JOptionPane.showConfirmDialog(AutoAppFrame.this, "Do you want delete" + tableModel.getValueAt(selectedRow, 0));

if (result == JOptionPane.OK\_OPTION) {

tableModel.deleteRow(tableData.getSelectedRow());

tableModel.fireTableDataChanged();

}

}

}

});

updateBtn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int selectedRow = tableData.getSelectedRow();

if (selectedRow != -1) {

new DataFrame(tableModel, selectedRow);

} else {

JOptionPane.showMessageDialog(AutoAppFrame.this, "Not selected data");

}

}

});

newListBtn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try{

int result = JOptionPane.showConfirmDialog(AutoAppFrame.this, "All data from list will be cleared. Continue?");

if (result == JOptionPane.OK\_OPTION) {

tableModel.clearData();

tableModel.fireTableDataChanged();

}

} catch (Exception ex) {

System.out.println(ex);

}

}

});

loadBtn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try{

JFileChooser file = new JFileChooser();

int result = file.showDialog(AutoAppFrame.this, "Открыть файл");

if (result == JFileChooser.APPROVE\_OPTION) {

File fin = file.getSelectedFile();

tableModel.loadFromFile(fin);

tableModel.fireTableDataChanged();

}

} catch (Exception ex) {

System.out.println(ex);

}

}

});

saveBtn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try{

JFileChooser file = new JFileChooser();

int result = file.showSaveDialog(AutoAppFrame.this);

if (result == JFileChooser.APPROVE\_OPTION) {

File fout = file.getSelectedFile();

tableModel.saveToFile(fout);

}

} catch (Exception ex) {

System.out.println(ex.toString());

}

}

});

}

private void BuildTableField() {

tableModel = new AutoTableModel();

tableData = new JTable(tableModel);

JScrollPane scrollPane = new JScrollPane(tableData);

scrollPane.setBorder(new EmptyBorder(20,20,20,20));

scrollPane.setVisible(true);

contentPane.add(scrollPane,BorderLayout.CENTER);

}

}

“AutoTableModel.java”

import java.io.File;

import java.util.Iterator;

import java.util.LinkedList;

import javax.swing.table.AbstractTableModel;

@SuppressWarnings("serial")

public class AutoTableModel extends AbstractTableModel {

private AutoCollection \_data;

@Override

public void fireTableDataChanged() {

super.fireTableDataChanged();

sort();

}

public AutoTableModel() {

\_data = new AutoCollection();

}

public AutoTableModel(AutoCollection data) {

\_data = data;

}

public AutoTableModel(LinkedList<Auto> data) {

\_data = new AutoCollection(data);

}

public void loadFromFile(File file) {

\_data.LoadFromFile(file);

}

public void saveToFile(File file) {

\_data.SaveToFile(file);

}

public void sort() {

\_data.sort(null);

}

@Override

public int getColumnCount() {

return Auto.getDataFieldsCount();

}

@Override

public int getRowCount() {

return \_data.size();

}

@Override

public Object getValueAt(int row, int col) {

Auto entityInRow = \_data.get(row);

return entityInRow.getValueAtField(col);

}

public Class<?> getColumnClass(int columnIndex){

return getValueAt(0, columnIndex).getClass();

}

public String getColumnName(int colNum) {

return Auto.getFieldName(colNum);

}

@Override

public boolean isCellEditable(int row, int col) {

return false;

}

public void setValueAt(Object value, int rowIndex, int colIndex) {

Auto temp = \_data.get(rowIndex);

temp.setValueAtField(value, colIndex);

}

public boolean addRow(Auto entity) {

if (entity != null) {

\_data.add(entity);

int indexOfNewEntity = \_data.indexOf(entity);

return (indexOfNewEntity == -1) ? false : true;

}

return false;

}

public boolean deleteRow(int index) {

if (index < \_data.size()) {

\_data.remove(index);

return true;

} else {

return false;

}

}

public boolean removeRange(int[] indexes) {

if (indexes != null) {

for(int i = 0; i < indexes.length; ++i) {

\_data.remove(indexes[i] - i);

}

return true;

} else {

return false;

}

}

public void clearData() {

\_data.clear();

}

public boolean updateRow(int index, Auto entity) {

Auto result = \_data.set(index, entity);

return (result == null) ? false : true;

}

public boolean isExist(Auto entity) {

Iterator<Auto> tempIter = \_data.find(entity.getMark());

return tempIter == null ? false : true;

}

public Auto getRowEntity(int index) {

return \_data.get(index);

}

}

“DataFrame.java”

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.border.EmptyBorder;

public class DataFrame extends JFrame{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private AutoTableModel \_dataTable;

final static int width = 700;

final static int height = 150;

final static int layoutRowCount = 2;

final static int layoutColCount = 4;

JLabel labelForMarkField = new JLabel("Mark:");

JLabel labelForYearField = new JLabel("Produce year:");

JLabel labelForEnfineVolField = new JLabel("Engine volume");

JLabel labelForMaxSpeedField = new JLabel("Max speed");

JTextField markField = new JTextField();

JTextField yearField = new JTextField();

JTextField engineVolField = new JTextField();

JTextField maxSpeedField = new JTextField();

JButton btnOk = new JButton("Add");

JButton btnCancel = new JButton("Reset");

public DataFrame(AutoTableModel dataTable) {

try {

\_dataTable = dataTable;

InitFrame();

markField = new JTextField();

yearField = new JTextField();

engineVolField = new JTextField();

maxSpeedField = new JTextField();

BuildFields();

BuildButtons(-1);

}catch(Exception ex) {

JOptionPane.showMessageDialog(this, "Inner exception" + ex.getMessage());

}

}

public DataFrame(AutoTableModel dataTable, int index) {

try {

\_dataTable = dataTable;

InitFrame();

Auto entity = \_dataTable.getRowEntity(index);

markField = new JTextField(entity.getMark());

yearField = new JTextField(String.valueOf(entity.getProductionYear()));

engineVolField = new JTextField(String.valueOf(entity.getEngineVolume()));

maxSpeedField = new JTextField(String.valueOf(entity.getMaxSpeed()));

BuildFields();

BuildButtons(index);

}catch(Exception ex) {

JOptionPane.showMessageDialog(this, "Inner exception" + ex.getMessage());

}

}

private void InitFrame() {

this.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

this.setSize(width, height);

this.setVisible(true);

JPanel panel = new JPanel();

panel.setBorder(new EmptyBorder(15,15,15,15));

panel.setLayout(new BorderLayout());

this.setContentPane(panel);

}

private void BuildFields() {

try {

JPanel fieldPanel = new JPanel();

fieldPanel.setLayout(new GridLayout(layoutRowCount, layoutColCount));

// fieldPanel.setLayout(new BorderLayout(layoutRowCount, layoutColCount));

fieldPanel.add(labelForMarkField);

fieldPanel.add(labelForYearField);

fieldPanel.add(labelForEnfineVolField);

fieldPanel.add(labelForMaxSpeedField);

fieldPanel.add(markField);

fieldPanel.add(yearField);

fieldPanel.add(engineVolField);

fieldPanel.add(maxSpeedField);

this.add(fieldPanel, BorderLayout.CENTER);

} catch (Exception ex) {

throw new ExceptionInInitializerError();

}

}

private void BuildButtons(int index) {

try {

JPanel buttonPanel = new JPanel();

buttonPanel.setLayout(new GridLayout(2, 1));

btnOk.setEnabled(true);

btnCancel.setEnabled(true);

buttonPanel.add(btnOk);

buttonPanel.add(btnCancel);

btnOk.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

boolean result = false;

if (index == -1) {

result = \_dataTable.addRow(

new Auto(markField.getText(),

yearField.getText(),

engineVolField.getText(),

maxSpeedField.getText()

)

);

} else {

result = \_dataTable.updateRow(index, new Auto(markField.getText(),

yearField.getText(),

engineVolField.getText(),

maxSpeedField.getText()

));

DataFrame.this.dispose();

}

if (!result) {

return;

}

cleanAllFields();

} catch(NumberFormatException ex) {

JOptionPane.showMessageDialog(DataFrame.this, "Data not added, Fill all field or enter correctly format");

}

finally {

\_dataTable.fireTableDataChanged();

}

}

});

this.add(buttonPanel,BorderLayout.EAST);

btnCancel.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

cleanAllFields();

}

});

} catch (Exception ex) {

throw new ExceptionInInitializerError();

}

}

private void cleanAllFields() {

markField.setText("");

yearField.setText("");

engineVolField.setText("");

maxSpeedField.setText("");

}

}

“main.java”

import java.awt.EventQueue;

public class main {

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

AutoAppFrame frame = new AutoAppFrame();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

}