# ЛАБОРАТОРНАЯ РАБОТА №4

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

Работу выполнил: Мовенко Константин, ИС/б-21-2-о

Цель работы

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.

Задачи

Создать Java приложение с графическим интерфейсом пользователя, реализующее редактирование, сортировку и удаление данных заданного по варианту типа информации T (Таблица 1). 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 учесть следующие требования и рекомендации:

1. Создать публичный класс, представляющий заданный по варианту задания тип информации (т.е. строку таблицы);
2. Создать модель данных таблицы. Для этого создать класс, расширяющий абстрактный класс AbstractTableModel. Создать в нём объект коллекции типа T, соответствующий варианту задания.

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(<объект>)* – добавление элемента (строки);
  + *public void deleteRow(String Поле\_1)* – удаление элемента по значению поля 1;
  + *public void updateRow(int row, <объект>)* – изменение элемента заданной строки;

1. Для реализации окна приложения реализовать дочерний класс JFrame;
2. Для представления таблицы с данными использовать компонент класса JTable, размести в 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 JButtonJButton 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 должны происходить по ключевому полю P в направлении сортировки U (Таблица 1);

Вариант задания

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип информации | Поле для сортировки (P) | Направление (U) | Тип коллекции (T) |
| 13 | C | 1 | Убывание | HashSet |

Таблица 1 – Вариант задания

C: Автомобиль (марка, год выпуска, объём двигателя, максимальная скорость).

Ход работы

При запуске написанной программы на экране появилось новое рабочее окно программы (Рисунок 1).

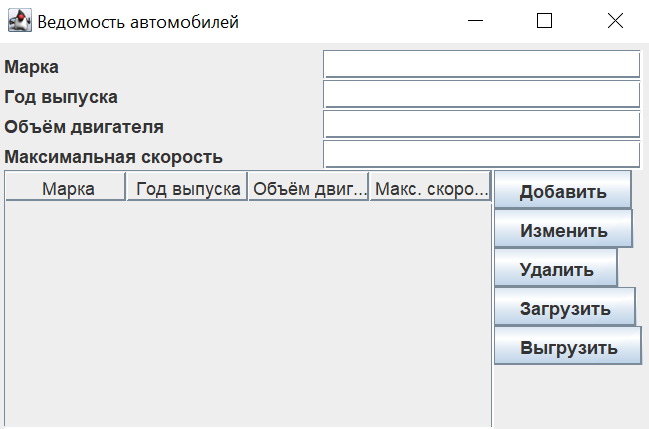


Рисунок 1 – Окно интерфейса программы

Был создан файл формата csv, содержащий информацию о нескольких автомобилях (Рисунок 2).

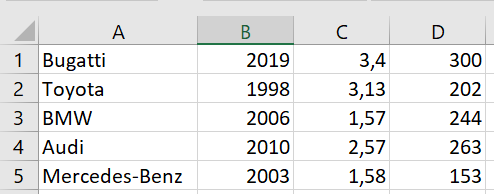


Рисунок 2 – Файл входных данных

При нажатии кнопки «Загрузить» и выборе созданного файла таблица программы была заполнена значениями из файла, данные были считаны верно (Рисунок 3).

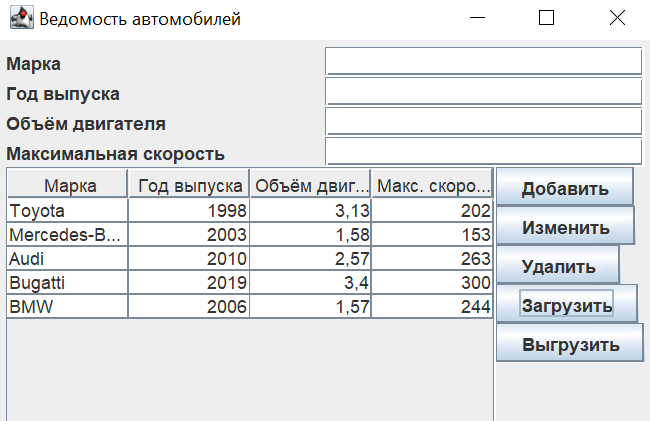


Рисунок 3 – Результат считывания данных из файла

Над полученной таблицей был проведён ряд операций: строка с «Audi» была удалена, в строке с Toyota год был заменён на 2000, была добавлена новая строка (Рисунок 4).

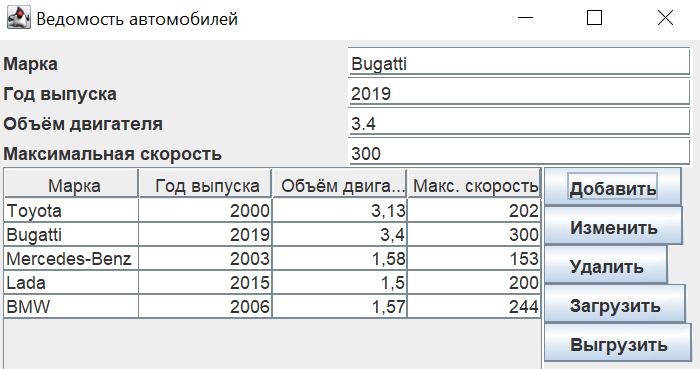


Рисунок 4 – Обновление таблицы

При нажатии кнопки «Выгрузить» данные полученной таблицы были выгружены в другой csv файл (Рисунок 5).

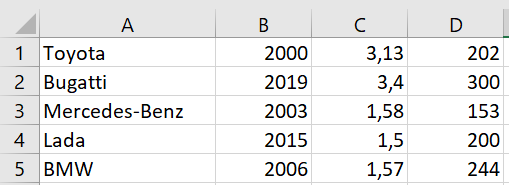


Рисунок 5 – Выгрузка данных в файл

Текст программы

Листинг 1 – jfMyFrame.java

**import** java.awt.BorderLayout;

**import** java.awt.EventQueue;

**import** java.awt.GridLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** java.io.File;

**import** javax.swing.BoxLayout;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.JTable;

**import** javax.swing.ListSelectionModel;

**import** javax.swing.border.EmptyBorder;

**public** **class** jfMyFrame **extends** JFrame

{

**private** **static** **final** **long** ***serialVersionUID*** = 26268648356920251L;

// Используемые компоненты

**private** **final** JPanel rootPane = **new** JPanel(); // главная (корневая) панель

**private** **final** AutoTableModel tableModel = **new** AutoTableModel(); // модель табличного представления данных

**private** **final** JTable dataTable = **new** JTable(tableModel); // таблица данных

**private** **final** JScrollPane scrollPane = **new** JScrollPane(dataTable); // прокручиваемая панель

**private** **final** JPanel buttonsPane = **new** JPanel(); // панель для кнопок

**private** **final** JPanel fieldsPane = **new** JPanel(); // панель для полей ввода

// Кнопки

**private** **final** JButton addButton = **new** JButton("Добавить");

**private** **final** JButton updateButton = **new** JButton("Изменить");

**private** **final** JButton deleteButton = **new** JButton("Удалить");

**private** **final** JButton inputButton = **new** JButton("Загрузить");

**private** **final** JButton outputButton = **new** JButton("Выгрузить");

// Подписи полей ввода

**private** **final** JLabel brandFieldLabel = **new** JLabel("Марка");

**private** **final** JLabel yearFieldLabel = **new** JLabel("Год выпуска");

**private** **final** JLabel engineVolumeFieldLabel = **new** JLabel("Объём двигателя");

**private** **final** JLabel maxSpeedFieldLabel = **new** JLabel("Максимальная скорость");

// Класс управления полями ввода данных

**private** **final** TextFields textFieldGroup = **new** TextFields();

// Запуск окна приложения

**public** **static** **void** main(String[] args)

{

EventQueue.*invokeLater*(**new** Runnable()

{

**public** **void** run ()

{

**try**

{

jfMyFrame frame = **new** jfMyFrame();

frame.setVisible(**true**);

}

**catch** (Exception e)

{ e.printStackTrace(); }

}

});

}

// Окно приложения

**public** jfMyFrame()

{

// Настройки окна приложения

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

setTitle("Ведомость автомобилей");

setBounds(200, 100, 450, 300);

// Настройки корневой панели

rootPane.setBorder(**new** EmptyBorder(5, 5, 5, 5));

rootPane.setLayout(**new** BorderLayout(0, 0));

setContentPane(rootPane);

// Настройки прокручеваемой панели

rootPane.add(scrollPane, BorderLayout.***CENTER***);

// Панель для кнопок

rootPane.add(buttonsPane, BorderLayout.***EAST***);

buttonsPane.setLayout(**new** BoxLayout(buttonsPane, BoxLayout.***Y\_AXIS***));

buttonsPane.add(addButton);

buttonsPane.add(updateButton);

buttonsPane.add(deleteButton);

buttonsPane.add(inputButton);

buttonsPane.add(outputButton);

// Панель с полями ввода

fieldsPane.setLayout(**new** GridLayout(0, 2));

rootPane.add(fieldsPane, BorderLayout.***NORTH***);

fieldsPane.add(brandFieldLabel);

fieldsPane.add(textFieldGroup.brandField);

fieldsPane.add(yearFieldLabel);

fieldsPane.add(textFieldGroup.yearField);

fieldsPane.add(engineVolumeFieldLabel);

fieldsPane.add(textFieldGroup.engineVolumeField);

fieldsPane.add(maxSpeedFieldLabel);

fieldsPane.add(textFieldGroup.maxSpeedField);

// Настройка таблицы данных

dataTable.getSelectionModel().setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);

// Действия при нажатии кнопки "Добавить"

addButton.addActionListener(**new** ActionListener()

{

**public** **void** actionPerformed(ActionEvent e)

{

**try**

{

tableModel.addRow(textFieldGroup.getObject());

}

**catch** (NumberFormatException ex)

{

JOptionPane.*showMessageDialog*(**null**, "ОШИБКА: Данные заполнены неверно");

}

}

});

// Действия при нажатии кнопки "Изменить"

updateButton.addActionListener(**new** ActionListener()

{

**public** **void** actionPerformed(ActionEvent e)

{

**try**

{

**if** (dataTable.getSelectedRow() >= 0)

{

tableModel.updateRow(dataTable.getSelectedRow(), textFieldGroup.getObject());

}

**else**

{

JOptionPane.*showMessageDialog*(**null**, "ОШИБКА: Не выбрана строка");

}

}

**catch** (NumberFormatException ex)

{

JOptionPane.*showMessageDialog*(**null**, "ОШИБКА: Данные заполнены неверно");

}

}

});

// Действия при нажатии кнопки "Удалить"

deleteButton.addActionListener(**new** ActionListener()

{

**public** **void** actionPerformed(ActionEvent e)

{

tableModel.deleteRow(textFieldGroup.brandField.getText());

}

});

// Действия при нажатии кнопки "Загрузить"

inputButton.addActionListener(**new** ActionListener()

{

**public** **void** actionPerformed(ActionEvent e)

{

**try**

{

File file = **new** FileChooser().getReadFile();

**if** (file != **null**)

{

tableModel.clearRows();

tableModel.addRows((**new** FileManager(file)).readSet());

}

}

**catch** (Exception ex)

{

JOptionPane.*showMessageDialog*(**null**, ex.getMessage());

}

}

});

// Действия при нажатии кнопки "Выгрузить"

outputButton.addActionListener(**new** ActionListener()

{

**public** **void** actionPerformed(ActionEvent e)

{

**try**

{

File file = **new** FileChooser().getSaveFile();

**if** (file != **null**)

{

**new** FileManager(file).writeSet(tableModel.getIterator());

}

}

**catch** (Exception ex)

{

JOptionPane.*showMessageDialog*(**null**, ex.getMessage());

}

}

});

// Действия при нажатии на строку таблицы

dataTable.addMouseListener(**new** MouseAdapter()

{

@Override

**public** **void** mouseClicked(MouseEvent event)

{

textFieldGroup.fillTextFields(tableModel.getRow(dataTable.getSelectedRow()));

}

});

}

}

Листинг 2 – Auto.java

// Класс для типа информации C по варианту

**public** **class** Auto

{

// Параметры

**public** String brand; // марка

**public** **int** year; // год выпуска

**public** **float** engineVolume; // объём двигателя, л.

**public** **int** maxSpeed; // макс. скорость, км/ч.

// Передача параметров через конструктор

**public** Auto(String brand, **int** year, **float** engineVolume, **int** maxSpeed)

{

**this**.brand = brand;

**this**.year = year;

**this**.engineVolume = engineVolume;

**this**.maxSpeed = maxSpeed;

}

@Override

**public** String toString() // представление объекта в виде строки (для csv файла)

{

**return** String.*format*("%s;%d;%f;%d", brand, year, engineVolume, maxSpeed);

}

@Override

**public** **boolean** equals(Object obj) // сравнение с другим объектом

{

**if** (obj **instanceof** Auto)

{

Auto other = (Auto) obj;

**return** brand == other.brand && year == other.year && maxSpeed == other.maxSpeed && engineVolume == other.engineVolume;

}

**return** **false**;

}

}

Листинг 3 – AutoTableModel.java

**import** java.util.Collection;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** javax.swing.table.AbstractTableModel;

// Модель табличного представления множества объектов класса Auto

**public** **class** AutoTableModel **extends** AbstractTableModel

{

**private** **static** **final** **long** ***serialVersionUID*** = 6181788579049573897L;

**private** **final** HashSet<Auto> data = **new** HashSet<Auto>();

**private** **final** String[] header = { "Марка", "Год выпуска", "Объём двигателя", "Макс. скорость" };

@Override

**public** **int** getRowCount() // возвращает число строк

{ **return** data.size(); }

@Override

**public** **int** getColumnCount() // возвращает число столбцов

{ **return** 4; }

@Override

**public** Class<?> getColumnClass(**int** columnIndex) // возвращает тип значения столбца

{

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

}

@Override

**public** Object getValueAt(**int** rowIndex, **int** columnIndex) // возвращает значение поля таблицы

{

Auto element = getRow(rowIndex);

**switch** (columnIndex)

{

**case** 0: **return** element.brand;

**case** 1: **return** element.year;

**case** 2: **return** element.engineVolume;

**case** 3: **return** element.maxSpeed;

}

**return** **null**;

}

@Override

**public** **void** setValueAt(Object value, **int** rowIndex, **int** columnIndex) // выполняет изменение данных в ячейке

{

Auto element = getRow(rowIndex);

**switch** (columnIndex)

{

**case** 0: element.brand = (String)value; **break**;

**case** 1: element.year = (**int**)value; **break**;

**case** 2: element.engineVolume = (**float**)value; **break**;

**case** 3: element.maxSpeed = (**int**)value; **break**;

}

}

@Override

**public** **boolean** isCellEditable(**int** rowIndex, **int** columnIndex) // узнать возможность редактирования ячейки

{ **return** **true**; }

// Возвращает имя столбца

**public** String getColumnName(**int** columnIndex)

{

**return** header[columnIndex];

}

**public** Auto getRow(**int** rowIndex) // возвращает объект по индексу

{

**int** counter = 0;

**for** (Auto obj: data)

{

**if** (counter++ == rowIndex)

{

**return** obj;

}

}

**return** **null**;

}

// Возвращает итератор коллекции

**public** Iterator<Auto> getIterator()

{ **return** data.iterator(); }

// Добавляет к таблице объект

**public** **void** addRow(Auto obj)

{

data.add(obj);

fireTableDataChanged();

}

// Добавляет в таблицу объекты коллекции

**public** **void** addRows(Collection<Auto> rows)

{

rows.forEach(data::add);

fireTableDataChanged();

}

// Удаляет объект по значению поля 1

**public** **void** deleteRow(String brand)

{

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

**while** (iter.hasNext())

{

**if** (iter.next().brand.equals(brand))

{

iter.remove();

//break;

}

}

fireTableDataChanged();

**return**;

}

// Обновление строки в таблице

**public** **void** updateRow(**int** rowIndex, Auto obj)

{

setValueAt(obj.brand, rowIndex, 0);

setValueAt(obj.year, rowIndex, 1);

setValueAt(obj.engineVolume, rowIndex, 2);

setValueAt(obj.maxSpeed, rowIndex, 3);

fireTableDataChanged();

}

// Полная очистка таблицы

**public** **void** clearRows()

{

data.clear();

fireTableDataChanged();

}

}

Листинг 4 – TextFields.java

**import** javax.swing.JTextField;

// Класс управления текстовыми полями ввода данных

**public** **class** TextFields

{

// Текстовые поля

**public** **final** JTextField brandField = **new** JTextField();

**public** **final** JTextField yearField = **new** JTextField();

**public** **final** JTextField engineVolumeField = **new** JTextField();

**public** **final** JTextField maxSpeedField = **new** JTextField();

// Построение объекта Auto по значениям полей ввода

**public** Auto getObject()

{

Auto obj = **new** Auto(brandField.getText(), Integer.*parseInt*(yearField.getText()),

Float.*parseFloat*(engineVolumeField.getText()), Integer.*parseInt*(maxSpeedField.getText()));

**return** obj;

}

// Заполнение текстовых полей значениями свойств объекта

**public** **void** fillTextFields(Auto obj)

{

brandField.setText(obj.brand);

yearField.setText(Integer.*toString*(obj.year));

engineVolumeField.setText(Float.*toString*(obj.engineVolume));

maxSpeedField.setText(Integer.*toString*(obj.maxSpeed));

}

// Очистка всех текстовх полей

**public** **void** clearAll()

{

brandField.setText("");

yearField.setText("");

engineVolumeField.setText("");

maxSpeedField.setText("");

}

}

Листинг 5 – FileManager.java

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.util.HashSet;

**import** java.util.Iterator;

// Класс для работы с файловым вводом-выводом

**public** **class** FileManager

{

**private** File file = **null**; // выбранный файл

**public** FileManager(File file) // конструктор

{

**this**.file = file;

}

// Считывание данных из файла

**public** HashSet<Auto> readSet() **throws** IOException

{

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

{

HashSet<Auto> data = **new** HashSet<Auto>();

String line;

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

{

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

params[2] = params[2].replace(",", ".");

Auto car = **new** Auto(params[0], Integer.*parseInt*(params[1]), Float.*parseFloat*(params[2]),

Integer.*parseInt*(params[3]));

data.add(car);

}

reader.close();

**return** data;

}

**catch** (NumberFormatException | ArrayIndexOutOfBoundsException e)

{

**throw** **new** IllegalArgumentException("Ошибка считывания файла ввода");

}

}

// Запись данных в файл

**public** **void** writeSet(Iterator<Auto> collection) **throws** IOException

{

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

{

**while** (collection.hasNext())

{

Auto auto = collection.next();

writer.write(auto.toString() + '\n');

}

}

}

}

Листинг 6 – FileChooser.java

**import** java.io.File;

**import** javax.swing.JFileChooser;

**import** javax.swing.filechooser.FileNameExtensionFilter;

// Класс для окна выбора файла для ввода-вывода файла

**public** **class** FileChooser

{

**private** **final** JFileChooser chooser; // компонент выбора файла

**public** FileChooser()

{

chooser = **new** JFileChooser();

chooser.setFileFilter(**new** FileNameExtensionFilter("Файл CSV", "csv"));

}

// Выбрать файл, из которого будут загружены данные

**public** File getReadFile()

{

chooser.setDialogTitle("Открыть файл");

**int** result = chooser.showOpenDialog(**null**);

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

{

**return** chooser.getSelectedFile();

}

**else**

{ **return** **null**; }

}

// Выбрать файл для сохранения данных

**public** File getSaveFile()

{

chooser.setDialogTitle("Сохранить");

**int** result = chooser.showSaveDialog(**null**);

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

{

**return** chooser.getSelectedFile();

}

**else**

{ **return** **null**; }

}

}

Вывод

В ходе работы было проведено ознакомление с библиотекой графического интерфейса Swing и её компонентами. Были приобретены навыки разработки программ с графическим интерфейсом с помощью плагина WindowBuilder.

Полученные знания позволяют создавать приложения с простейшим пользовательским интерфейсом, что даёт пользователю удобный способ взаимодействия с программой через графические элементы.