**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторно-практической работе № 2-5**

**по дисциплине «Объектно - ориентированное программирование на языке Java»**

Тема: **«Объектно - ориентированное программирование на языке Java»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Шарпинский Д. А. |  |
| Преподаватель | Павловский М.Г |  |

Санкт-Петербург

2024

**Цель работы**

1. **Знакомство с правилами построения экранной формы.**
2. **Обработка событий.**
3. **Обработка исключений.**
4. **Сохранение и загрузка данных из файла.**

**Описание задания:**

**-**

**Назначение экранной формы:**

Экранная форма представляет собой интерфейс для менеджера соревнований автогонщиков. В форме можно управлять данными о командах, гонщиках, трассах и соревнованиях. Пользователь может просматривать информацию в табличном виде, выполнять поиск по различным критериям и сохранять изменения.

* Вводимая информация:
* Поиск: Пользователь может вводить текстовый запрос в поле для поиска и выбирать категорию для фильтрации (например, "Команды", "Гонщики", "Трассы", "Соревнования"), полностью не реализован.
* Сохранение данных: при необходимости пользователь может сохранить изменения в данных, хотя кнопка для сохранения пока не реализована.
* Выводимая информация:

1. Команды: Таблица с командами автогонщиков. Выводятся колонки:
   * Название команды
   * Страна, представляющая команду
2. Гонщики: Таблица с гонщиками. Выводятся колонки:
   * Имя гонщика
   * Команда, за которую он выступает
   * Количество очков, набранных гонщиком
3. Трассы: Таблица с трассами, на которых проходят гонки. Выводятся колонки:
   * Название трассы
   * Местоположение трассы (страна)
4. Соревнования: Таблица с информацией о соревнованиях. Выводятся колонки:
   * Название трассы
   * Дата проведения соревнования
   * Время начала заезда

* Функции, доступные пользователю:

1. Просмотр данных: Пользователь может переключаться между вкладками и просматривать данные по командам, гонщикам, трассам и соревнованиям в табличном формате.
2. Поиск: Пользователь может выполнить поиск по заданным критериям (например, найти команду, гонщика или трассу). Для этого нужно выбрать категорию (команды, гонщики, трассы или соревнования) и ввести запрос в текстовое поле, но полностью не реализован.
3. Сохранение изменений: Пользователь может сохранить внесённые изменения в базу данных с помощью кнопки "Сохранить", хотя в текущей версии программы функционал сохранения не реализован.

* Преимущества:
* Удобный интерфейс для работы с четырьмя категориями данных.
* Возможность поиска информации по введённым критериям.
* Интуитивно понятный дизайн с разделением данных по вкладкам.

**Внешний вид экранной формы:**

Спроектировано в Figma

Референс:

Изображение выглядит как снимок экрана, гаджет, Электронное устройство, мультимедиа

Автоматически созданное описание

Готовый результат:

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

**Описание экранной формы**

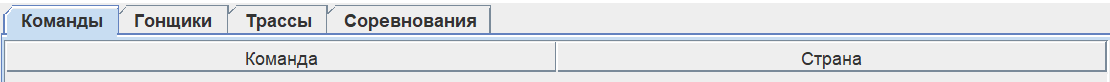
**1.** Панель инструментов в Java была реализована с помощью компонента JToolBar из библиотеки Swing. JToolBar представляет собой панель, которая может содержать различные компоненты, такие как кнопки, текстовые поля, выпадающие списки и другие элементы управления. Обычно панель инструментов используется для размещения часто используемых функций, таких как кнопки для сохранения, открытия или удаления файлов.

Изображение выглядит как снимок экрана

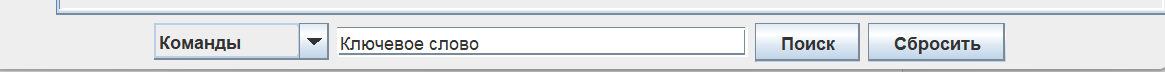
Автоматически созданное описание

**2.** в основной части сверху располагаются вкладки и под ними столбцы.

Для реализации вкладок использовался компонент JTabbedPane из библиотеки Swing. Этот компонент позволяет создать интерфейс с несколькими вкладками, каждая из которых может содержать различные панели с элементами управления, такими как таблицы, кнопки и текстовые поля.



3.в нижней расположены поиск по вкладке и сброс этой вкладки до первоначального значения. поиск и сброс, были реализованы с использованием компонентов **JTextField** для ввода поискового запроса и **JButton** для выполнения действий поиска и сброса



**Описание действий реализованными слушателями**

Добавление данных (addButton.addActionListener)

Слушатель для кнопки "Добавить" позволяет создать новую запись в таблице.

* Слушатель события: при нажатии на кнопку "Добавить":
  + Открывает интерфейс для ввода новой записи (возможно, форма для заполнения данных).
  + Добавляет новые данные в соответствующую таблицу.
  + Обновляет таблицу для отображения новой строки.
* Действия:
  + Открывает форму для ввода данных.
  + Получает данные от пользователя.
  + Добавляет новую запись в таблицу.
  + Обновляет интерфейс таблицы для отображения новой строки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, Параллельный

Автоматически созданное описание

**Удаление данных (deleteButton.addActionListener)**

Слушатель для кнопки "Удалить" позволяет удалить выбранные строки из таблицы.

* Слушатель события: при нажатии на кнопку "Удалить":
  + Проверяет, какие строки выбраны в таблице.
  + Удаляет выбранные строки из модели таблицы.
  + Обновляет таблицу, убирая удалённые строки.
* Действия:
  + Получает список выбранных строк.
  + Удаляет выбранные строки из модели таблицы.
  + Обновляет интерфейс для отображения оставшихся данных.

Изображение выглядит как текст, снимок экрана, число, Параллельный

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, Параллельный

Автоматически созданное описание

**Сохранение данных (saveButton.addActionListener)**

Слушатель для кнопки "Сохранить" позволяет сохранить новые или изменённые данные в таблицу.

* Слушатель события: При нажатии на кнопку "Сохранить":
  + Считывает данные из текстовых полей ввода.
  + Определяет, в какой таблице нужно обновить данные (команды, гонщики и т.д.).
  + Либо добавляет новую строку в таблицу, либо обновляет существующую в зависимости от контекста (создание нового элемента или редактирование существующего).
* Действия:
  + Получает данные из текстовых полей.
  + Определяет текущую вкладку.
  + Вставляет данные в соответствующую таблицу.
  + Обновляет модель таблицы и отображает новые данные.

Изображение выглядит как текст, программное обеспечение, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

**Сброс и поиск (resetButton.addActionListener),** **(filter.addActionListener)**

Слушатель для кнопки сброса восстанавливает исходные данные таблицы после фильтрации.

* Слушатель события: при нажатии на кнопку "Сброс":
  + Определяет, какая таблица была отфильтрована.
  + Восстанавливает исходные данные из сохранённой копии.
  + Очищает фильтр поиска и восстанавливает отображение всех строк таблицы.
* Действия:
  + Определяет, какую таблицу нужно сбросить.
  + Восстанавливает исходные данные.
  + Очищает текущее состояние таблицы и загружает в неё исходные данные.

Изображение выглядит как текст, снимок экрана, число, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, Параллельный

Автоматически созданное описание

**Перечень ситуаций, которые контролируются с помощью исключений**

* **Пустые поля для ввода**
* **Ситуация:** Пользователь нажимает кнопку авторизации, не заполнив поля "Логин" или "Пароль".
* **Контроль с помощью исключений:**
  + При нажатии на кнопку, программа проверяет, что поля для ввода логина и пароля заполнены.
  + Если одно из полей пустое, выбрасывается исключение, например, IllegalArgumentException с сообщением об ошибке "Логин и пароль не могут быть пустыми".
* **Решение:** Исключение перехватывается, и пользователю выводится сообщение о том, что нужно заполнить все поля.
* **2. Некорректный формат логина**
* **Ситуация:** Пользователь вводит логин в неверном формате (например, слишком короткий, содержит недопустимые символы или не соответствует формату электронной почты).
* **Контроль с помощью исключений:**
  + Перед отправкой данных проверяется формат введённого логина. Например, можно проверить, соответствует ли логин формату email (если это необходимо).
  + Если формат логина не соответствует требованиям, выбрасывается исключение InvalidFormatException или подобное.
* **Решение:** Исключение перехватывается, и пользователю выводится сообщение, указывающее на неверный формат логина.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Текст программы:**

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

/\*\*

\* Класс {@code BookList} представляет основное окно приложения для управления списком книг.

\* Приложение позволяет управлять списками команд, гонщиков, трасс и соревнований.

\* Предоставляется функциональность для добавления, удаления, сохранения данных, а также поиска и регистрации пользователей.

\*/

public class BookList {

// Объявления графических компонентов

private JFrame bookList; // Основное окно приложения

private DefaultTableModel modelTeams; // Модель таблицы для команд

private DefaultTableModel modelDrivers; // Модель таблицы для гонщиков

private DefaultTableModel modelTracks; // Модель таблицы для трасс

private DefaultTableModel modelSorev; // Модель таблицы для соревнований

private JButton save; // Кнопка для сохранения данных

private JButton addFile; // Кнопка для добавления файла

private JButton delFile; // Кнопка для удаления строки

private JButton Trash; // Кнопка для удаления всех данных из вкладки

private JButton register; // Кнопка для регистрации пользователя

private JScrollPane scrollTeams; // Прокручиваемая панель для таблицы команд

private JScrollPane scrollDrivers; // Прокручиваемая панель для таблицы гонщиков

private JScrollPane scrollTracks; // Прокручиваемая панель для таблицы трасс

private JScrollPane scrollSorev; // Прокручиваемая панель для таблицы соревнований

private JTable tableTeams; // Таблица для команд

private JTable tableDrivers; // Таблица для гонщиков

private JTable tableTracks; // Таблица для трасс

private JTable tableSorev; // Таблица для соревнований

private JComboBox<String> searchCriteria; // Компонент для выбора критерия поиска

private JTextField searchField; // Поле для ввода ключевого слова для поиска

private JButton filter; // Кнопка для выполнения поиска

protected String fileName; // Имя файла

private JButton resetButton; // Кнопка для сброса поиска

private String[][] originalTeamsData; // Исходные данные для таблицы команд

private String[][] originalDriversData; // Исходные данные для таблицы гонщиков

private String[][] originalTracksData; // Исходные данные для таблицы трасс

private String[][] originalSorevData; // Исходные данные для таблицы соревнований

/\*\*

\* Метод {@code show()} отображает главное окно приложения.

\* В этом методе происходит:

\* <ul>

\* <li>Создание и настройка главного окна приложения.</li>

\* <li>Создание кнопок управления и их размещение на панели инструментов.</li>

\* <li>Инициализация таблиц для отображения данных команд, гонщиков, трасс и соревнований.</li>

\* <li>Настройка вкладок для переключения между разными таблицами.</li>

\* <li>Создание панели поиска и добавление соответствующих обработчиков событий.</li>

\* </ul>

\* Этот метод должен быть вызван для отображения и инициализации интерфейса приложения.

\*/

public void show() {

// Создание окна

bookList = new JFrame("Book List");

bookList.setSize(800, 500);

bookList.setMinimumSize(new Dimension(800, 500));

bookList.setLocation(250, 80);

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

// Создание кнопок и прикрепление иконок

save = new JButton(new ImageIcon(getClass().getResource("/img/save.png")));

save.setPreferredSize(new Dimension(40, 30));

save.setToolTipText("Сохранить изменения из вкладки");

addFile = new JButton(new ImageIcon(getClass().getResource("/img/plus.png")));

addFile.setPreferredSize(new Dimension(40, 30));

addFile.setToolTipText("Добавить файл во вкладку");

delFile = new JButton(new ImageIcon(getClass().getResource("/img/minus.png")));

delFile.setPreferredSize(new Dimension(40, 30));

delFile.setToolTipText("Удалить строку из вкладки");

Trash = new JButton(new ImageIcon(getClass().getResource("/img/Trash.png")));

Trash.setPreferredSize(new Dimension(40, 30));

Trash.setToolTipText("Удаление всей вкладки");

// Новая кнопка для регистрации пользователя

register = new JButton(new ImageIcon(getClass().getResource("/img/User.png")));

register.setPreferredSize(new Dimension(40, 30));

register.setToolTipText("Регистрация пользователя");

// Добавление кнопок на панель инструментов

JToolBar toolBar = new JToolBar(JToolBar.VERTICAL); // Вертикальная ориентация

toolBar.setLayout(new BoxLayout(toolBar, BoxLayout.Y\_AXIS)); // Вертикальное расположение элементов

toolBar.add(save);

toolBar.add(addFile);

toolBar.add(delFile);

toolBar.add(Trash);

toolBar.add(register); // Добавляем кнопку регистрации

// Размещение панели инструментов

bookList.setLayout(new BorderLayout());

bookList.add(toolBar, BorderLayout.WEST);

// Создание таблиц и вкладок (команды, гонщики, трассы, соревнования)

String[] columnsTeams = {"Команда", "Страна"};

modelTeams = new DefaultTableModel(new String[][]{}, columnsTeams);

tableTeams = new JTable(modelTeams);

scrollTeams = new JScrollPane(tableTeams);

String[] columnsDrivers = {"Гонщик", "Команда", "Очки"};

modelDrivers = new DefaultTableModel(new String[][]{}, columnsDrivers);

tableDrivers = new JTable(modelDrivers);

scrollDrivers = new JScrollPane(tableDrivers);

String[] columnsTracks = {"Трасса", "Местоположение"};

modelTracks = new DefaultTableModel(new String[][]{}, columnsTracks);

tableTracks = new JTable(modelTracks);

scrollTracks = new JScrollPane(tableTracks);

String[] columnsSorev = {"Трасса", "Дата проведения", "Начало заезда"};

modelSorev = new DefaultTableModel(new String[][]{}, columnsSorev);

tableSorev = new JTable(modelSorev);

scrollSorev = new JScrollPane(tableSorev);

JTabbedPane navBar = new JTabbedPane();

navBar.addTab("Команды", scrollTeams);

navBar.addTab("Гонщики", scrollDrivers);

navBar.addTab("Трассы", scrollTracks);

navBar.addTab("Соревнования", scrollSorev);

bookList.add(navBar, BorderLayout.CENTER);

// Панель поиска

searchCriteria = new JComboBox<>(new String[]{"Команды", "Гонщики", "Трассы", "Соревнования"});

searchField = new JTextField("Поиск...", 30);

filter = new JButton("Поиск");

JPanel searchPanel = new JPanel();

searchPanel.add(searchCriteria);

searchPanel.add(searchField);

searchPanel.add(filter);

resetButton = new JButton("Сбросить");

searchPanel.add(resetButton);

bookList.add(searchPanel, BorderLayout.SOUTH);

// Обработчик события для кнопки сброса поиска

resetButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

resetSearch(); // Восстанавливаем исходные данные для активной вкладки

}

});

// Обработчик события для кнопки регистрации

register.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

showRegistrationDialog(); // Показать окно регистрации

}

});

// Обработчик события для кнопки поиска

filter.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

performSearch(); // Выполняем поиск по ключевому слову

}

});

searchField.addFocusListener(new FocusAdapter() {

@Override

public void focusGained(FocusEvent e) {

// Очистка поля, если текст равен placeholder

if (searchField.getText().equals("Поиск...")) {

searchField.setText("");

}

}

@Override

public void focusLost(FocusEvent e) {

// Восстановление placeholder, если поле пустое при потере фокуса

if (searchField.getText().isEmpty()) {

searchField.setText("Поиск...");

}

}

});

// Добавление данных в библиотеку

addFile.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

FileDialog addFileDialog = new FileDialog(bookList, "Добавление данных", FileDialog.LOAD);

addFileDialog.setVisible(true); // Отобразить запрос пользователю

String directory = addFileDialog.getDirectory();

String selectedFile = addFileDialog.getFile();

if (directory == null || selectedFile == null) {

return; // Если пользователь нажал «отмена», прекратить выполнение

}

String fileName = directory + selectedFile;

int selectedTab = navBar.getSelectedIndex(); // Получаем индекс активной вкладки

DefaultTableModel selectedModel;

switch (selectedTab) {

case 0: selectedModel = modelTeams; break;

case 1: selectedModel = modelDrivers; break;

case 2: selectedModel = modelTracks; break;

case 3: selectedModel = modelSorev; break;

default: return;

}

// Чтение и добавление данных в выбранную таблицу

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

String line;

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

String[] fields = line.split(","); // Укажите разделитель для CSV

selectedModel.addRow(fields); // Добавление строки в таблицу

}

} catch (IOException e) {

e.printStackTrace();

}

}

});

// Удаление данных вкладки

Trash.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

int selectedTab = navBar.getSelectedIndex(); // Получаем индекс активной вкладки

// Определяем выбранную таблицу

DefaultTableModel selectedModel;

String[][] originalData = null;

switch (selectedTab) {

case 0:

selectedModel = modelTeams;

originalData = originalTeamsData;

break;

case 1:

selectedModel = modelDrivers;

originalData = originalDriversData;

break;

case 2:

selectedModel = modelTracks;

originalData = originalTracksData;

break;

case 3:

selectedModel = modelSorev;

originalData = originalSorevData;

break;

default:

return; // Если выбран неизвестный индекс, выходим

}

// Очистка таблицы

selectedModel.setRowCount(0);

// Очищаем также оригинальные данные

if (originalData != null) {

originalData = new String[0][0];

// Обновляем ссылку на массив данных

switch (selectedTab) {

case 0: originalTeamsData = originalData; break;

case 1: originalDriversData = originalData; break;

case 2: originalTracksData = originalData; break;

case 3: originalSorevData = originalData; break;

}

}

}

});

// Сохранение файла

save.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

int selectedTab = navBar.getSelectedIndex(); // Получаем индекс активной вкладки

// Определяем выбранную таблицу и имя файла

DefaultTableModel selectedModel;

String fileName;

switch (selectedTab) {

case 0:

selectedModel = modelTeams;

fileName = "teams.csv";

break;

case 1:

selectedModel = modelDrivers;

fileName = "drivers.csv";

break;

case 2:

selectedModel = modelTracks;

fileName = "tracks.csv";

break;

case 3:

selectedModel = modelSorev;

fileName = "competitions.csv";

break;

default:

return; // Если выбран неизвестный индекс, выходим

}

// Диалоговое окно для сохранения файла

FileDialog saveDialog = new FileDialog(bookList, "Сохранение данных", FileDialog.SAVE);

saveDialog.setFile(fileName); // Используем имя файла на основе текущей вкладки

saveDialog.setVisible(true);

String directory = saveDialog.getDirectory();

String selectedFile = saveDialog.getFile();

if (directory == null || selectedFile == null) {

return; // Если пользователь нажал «отмена»

}

fileName = directory + selectedFile;

// Сохранение данных в выбранный файл

try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {

for (int i = 0; i < selectedModel.getRowCount(); i++) {

for (int j = 0; j < selectedModel.getColumnCount(); j++) {

writer.write((String) selectedModel.getValueAt(i, j)); // Запись значения ячейки

if (j < selectedModel.getColumnCount() - 1) {

writer.write(","); // Разделитель

}

}

writer.write("\n"); // Новая строка

}

} catch (IOException e) {

e.printStackTrace();

}

}

});

// Удаление строки из вкладки

delFile.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

// Получение активной вкладки

int selectedTab = navBar.getSelectedIndex();

JTable selectedTable;

DefaultTableModel selectedModel;

String[][] originalData = null;

// Определяем, какая таблица активна

switch (selectedTab) {

case 0:

selectedTable = tableTeams;

selectedModel = modelTeams;

originalData = originalTeamsData;

break;

case 1:

selectedTable = tableDrivers;

selectedModel = modelDrivers;

originalData = originalDriversData;

break;

case 2:

selectedTable = tableTracks;

selectedModel = modelTracks;

originalData = originalTracksData;

break;

case 3:

selectedTable = tableSorev;

selectedModel = modelSorev;

originalData = originalSorevData;

break;

default:

return;

}

// Получаем индекс выбранной строки в активной таблице

int selectedRow = selectedTable.getSelectedRow();

// Проверяем, выбрана ли строка

if (selectedRow != -1) {

// Удаление из таблицы

selectedModel.removeRow(selectedRow);

// Удаление из массива исходных данных, если он не пустой

if (originalData != null) {

originalData = removeRowFromOriginalData(originalData, selectedRow);

// Обновляем ссылку на массив данных

switch (selectedTab) {

case 0: originalTeamsData = originalData; break;

case 1: originalDriversData = originalData; break;

case 2: originalTracksData = originalData; break;

case 3: originalSorevData = originalData; break;

}

}

} else {

// Если строка не выбрана, показываем предупреждение

JOptionPane.showMessageDialog(bookList, "Выберите строку для удаления.");

}

}

});

// Включаем сортировку для таблиц

enableSorting();

// Отображаем основное окно

bookList.setVisible(true);

}

/\*\*

\* Удаляет строку из двумерного массива данных на основе заданного индекса.

\*

\* @param originalData Двумерный массив строк, представляющий исходные данные таблицы.

\* @param rowToRemove Индекс строки, которую необходимо удалить из массива.

\* @return Новый двумерный массив строк, содержащий все строки, кроме удаленной. Если массив изначально содержит

\* только одну строку, возвращается пустой массив.

\*/

private String[][] removeRowFromOriginalData(String[][] originalData, int rowToRemove) {

if (originalData.length <= 1) return new String[0][0]; // Если в массиве всего одна строка, возвращаем пустой массив

String[][] newData = new String[originalData.length - 1][originalData[0].length];

int newIndex = 0;

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

if (i == rowToRemove) continue; // Пропускаем удаляемую строку

newData[newIndex++] = originalData[i];

}

return newData;

}

/\*\*

\* Извлекает данные из переданной модели таблицы.

\* Этот метод используется для получения копии данных таблицы.

\*

\* @param model Модель таблицы, из которой извлекаются данные.

\* @return Двумерный массив строк, представляющий данные таблицы.

\*/

private String[][] getTableData(DefaultTableModel model) {

int rowCount = model.getRowCount();

int columnCount = model.getColumnCount();

String[][] tableData = new String[rowCount][columnCount];

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

for (int j = 0; j < columnCount; j++) {

tableData[i][j] = (String) model.getValueAt(i, j);

}

}

return tableData;

}

/\*\*

\* Включает возможность сортировки строк в таблицах.

\* Применяется для всех таблиц, добавляя в них сортировку по умолчанию.

\*/

private void enableSorting() {

tableTeams.setAutoCreateRowSorter(true);

tableDrivers.setAutoCreateRowSorter(true);

tableTracks.setAutoCreateRowSorter(true);

tableSorev.setAutoCreateRowSorter(true);

}

/\*\*

\* Сбрасывает поиск и восстанавливает исходные данные для выбранной вкладки.

\* Используется для восстановления состояния таблицы до состояния до поиска.

\*/

private void resetSearch() {

int selectedIndex = searchCriteria.getSelectedIndex(); // Определяем вкладку для сброса

DefaultTableModel model;

String[][] originalData;

switch (selectedIndex) {

case 0:

model = modelTeams;

originalData = originalTeamsData;

break;

case 1:

model = modelDrivers;

originalData = originalDriversData;

break;

case 2:

model = modelTracks;

originalData = originalTracksData;

break;

case 3:

model = modelSorev;

originalData = originalSorevData;

break;

default:

return; // Если выбран неизвестный индекс, выходим

}

// Восстанавливаем исходные данные таблицы

if (originalData != null) {

model.setRowCount(0); // Очищаем текущие данные

for (String[] row : originalData) {

model.addRow(row); // Восстанавливаем исходные данные

}

}

}

/\*\*

\* Выполняет поиск по ключевому слову в выбранной таблице.

\* Фильтрует строки таблицы на основе введенного ключевого слова.

\*/

private void performSearch() {

String keyword = searchField.getText().trim().toLowerCase(); // Получаем ключевое слово для поиска и приводим к нижнему регистру

int selectedIndex = searchCriteria.getSelectedIndex(); // Определяем выбранный критерий (вкладку для поиска)

DefaultTableModel model;

String[][] originalData;

switch (selectedIndex) {

case 0:

model = modelTeams;

if (originalTeamsData == null) { // Сохраняем данные только при первом поиске

originalTeamsData = getTableData(model);

}

originalData = originalTeamsData;

break;

case 1:

model = modelDrivers;

if (originalDriversData == null) {

originalDriversData = getTableData(model);

}

originalData = originalDriversData;

break;

case 2:

model = modelTracks;

if (originalTracksData == null) {

originalTracksData = getTableData(model);

}

originalData = originalTracksData;

break;

case 3:

model = modelSorev;

if (originalSorevData == null) {

originalSorevData = getTableData(model);

}

originalData = originalSorevData;

break;

default:

return; // Если выбран неизвестный индекс, выходим

}

// Очистим текущие данные таблицы

model.setRowCount(0);

// Фильтрация данных: ищем строки, которые содержат ключевое слово

for (String[] row : originalData) {

boolean match = false;

for (String cell : row) {

if (cell.toLowerCase().contains(keyword)) { // Ищем вхождение ключевого слова

match = true;

break;

}

}

if (match) {

model.addRow(row); // Добавляем строки, которые соответствуют критерию поиска

}

}

}

/\*\*

\* Показывает диалоговое окно для регистрации пользователя.

\* Предоставляет поля для ввода имени и пароля и выполняет проверку введенных данных.

\*/

private void showRegistrationDialog() {

// Создание диалогового окна

JDialog registerDialog = new JDialog(bookList, "Регистрация пользователя", true);

registerDialog.setSize(300, 200);

registerDialog.setLayout(new BorderLayout());

// Создание панели для ввода данных

JPanel inputPanel = new JPanel();

inputPanel.setLayout(new GridLayout(4, 2)); // Установите GridLayout для ввода

JLabel nameLabel = new JLabel("Имя:");

JTextField nameField = new JTextField();

JLabel passwordLabel = new JLabel("Пароль:");

JPasswordField passwordField = new JPasswordField();

inputPanel.add(nameLabel);

inputPanel.add(nameField);

inputPanel.add(passwordLabel);

inputPanel.add(passwordField);

// Создание панели с кнопками

JPanel buttonPanel = new JPanel(); // Используйте новую панель для кнопок

JButton registerButton = new JButton("Регистрация");

JButton cancelButton = new JButton("Отмена");

// Установка размеров для кнопок

Dimension buttonSize = new Dimension(120, 30); // Укажите желаемые размеры кнопки

registerButton.setPreferredSize(buttonSize);

cancelButton.setPreferredSize(buttonSize);

buttonPanel.add(registerButton);

buttonPanel.add(cancelButton);

// Добавление панелей в диалог

registerDialog.add(inputPanel, BorderLayout.CENTER); // Ввод данных

registerDialog.add(buttonPanel, BorderLayout.SOUTH); // Кнопки внизу

// Обработчик кнопки "Зарегистрироваться"

registerButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String name = nameField.getText();

String password = new String(passwordField.getPassword());

// Проверка, что поля не пустые

if (name.isEmpty() || password.isEmpty()) {

JOptionPane.showMessageDialog(registerDialog, "Поля не должны быть пустыми.", "Ошибка", JOptionPane.ERROR\_MESSAGE);

return;

}

// Проверка, что имя состоит только из букв

if (!name.matches("[a-zA-Zа-яА-Я]+")) {

JOptionPane.showMessageDialog(registerDialog, "Имя должно содержать только буквы.", "Ошибка", JOptionPane.ERROR\_MESSAGE);

return;

}

// Успешная регистрация

JOptionPane.showMessageDialog(registerDialog, "Регистрация успешна!", "Информация", JOptionPane.INFORMATION\_MESSAGE);

registerDialog.dispose(); // Закрыть окно после успешной регистрации

try {

// Создаем директорию, если она не существует

File directory = new File("./src/csv");

if (!directory.exists()) {

directory.mkdir();

}

// Записываем данные в файл в директории ./csv

try (BufferedWriter writer = new BufferedWriter(new FileWriter("./src/csv/user.csv", true))) {

writer.write(name + "," + password);

writer.newLine();

}

} catch (IOException ex) {

ex.printStackTrace();

}

}

});

// Обработчик кнопки "Отмена"

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

registerDialog.dispose(); // Закрыть окно без действий

}

});

// Показать диалоговое окно

registerDialog.setLocationRelativeTo(bookList);

registerDialog.setVisible(true);

}

/\*\*

\* Основной метод программы. Запускает приложение, создавая и показывая главное окно.

\*

\* @param args Аргументы командной строки, передаваемые в программу (не используются).

\*/

public static void main(String[] args) {

// Запуск основного окна приложения

new BookList().show();

}

}

**Ссылки на Github и видеоотчёт:**

<https://github.com/Pyc-Nya/vuz2/tree/denis/oop/lab2-5>

<https://www.youtube.com/watch?v=CRNZOUAbFSI&ab_channel=%D0%94%D0%B5%D0%BD%D0%B8%D1%81%D0%A8%D0%B0%D1%80%D0%BF%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9>