**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторно-практической работе № 6**

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

Тема: **«Обработка XML-документов»**

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

Санкт-Петербург

2024

**Цель работы**

Знакомство с технологией обработки XML-документов и файлов.

**Описание задания**

1. Заменить обработчики записи в файл и чтения из файла для работы с XML-файлами

2. Загрузить данные из файла, изменить их, сохранить в новый файл. Проверить корректность работы, открыв новый файл.

**Работа с файлами**

Обработчики для addFile и save служат для загрузки и сохранения данных в файл соответственно. Они были изменены для работы с XML-документами.

**XML Файлы**

Представляют из себя древовидную структуру из тегов. В моем случае содержание может быть представлено следующим образом:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<data>

<teams>

<row>

<Команда>Team A</Команда>

<Страна>Country A</Страна>

</row>

<row>

<Команда>Team A</Команда>

<Страна>Country A</Страна>

</row>

</teams>

<drivers>

<row>

<Гонщик>Driver A</Гонщик>

<Команда>Team A</Команда>

<Очки>10</Очки>

</row>

<row>

<Гонщик>Driver A</Гонщик>

<Команда>Team A</Команда>

<Очки>10</Очки>

</row></drivers>

<tracks>

<row>

<Трасса>НИМАРИН</Трасса>

<Местоположение>USA</Местоположение>

</row>

<row>

<Трасса>Local Nevada</Трасса>

<Местоположение>USA</Местоположение>

</row>

</tracks>

<sorev>

<row>

<Трасса>Канада</Трасса>

<Дата>март</Дата>

<Начало>пять</Начало>

</row>

<row>

<Трасса>Local Nevada</Трасса>

<Дата>июнь</Дата>

<Начало>десять</Начало>

</row>

</sorev>

</data>

**Ссылки**

<https://github.com/Pyc-Nya/vuz2/tree/denis/oop/6>

<https://youtu.be/bAn4kKaiG8o>

**Исходный текст программы**

import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.\*;  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
import org.w3c.dom.Node;  
import org.w3c.dom.NodeList;  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.transform.Transformer;  
import javax.xml.transform.TransformerFactory;  
import javax.xml.transform.dom.DOMSource;  
import javax.xml.transform.stream.StreamResult;  
  
*/\*\*  
 \* Класс {@code BookList} представляет основное окно приложения для управления списком книг.  
 \* Этот класс включает в себя функции для отображения таблиц данных команд, водителей, трасс, соревнований и  
 \* поиска по ним, а также регистрацию пользователей.  
 \*   
 \* <p>Приложение предоставляет функции сортировки данных, выполнения поиска по ключевым словам и восстановления исходных данных после поиска.</p>  
 \*/*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 show() отображает главное окно программы, в котором содержатся таблицы с данными  
 \* о командах, гонщиках, трассах и соревнованиях. В окне также представлены кнопки управления  
 \* для добавления, удаления и сохранения данных, а также функция поиска и регистрации пользователей.  
 \*/* public void show() {  
 // Создание окна  
 bookList = new JFrame("DATA BASE");  
 bookList.setSize(800, 500);  
 bookList.setLocation(250, 80);  
 bookList.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  
 // Создание кнопок и прикрепление иконок  
 save = new JButton(new ImageIcon("./img/save.png"));  
 save.setPreferredSize(new Dimension(40, 30));  
 save.setToolTipText("Сохранить изменения из вкладки");  
  
 addFile = new JButton(new ImageIcon("./img/plus.png"));  
 addFile.setPreferredSize(new Dimension(40, 30));  
 addFile.setToolTipText("Добавить файл во вкладку");  
  
 delFile = new JButton(new ImageIcon("./img/minus.png"));  
 delFile.setPreferredSize(new Dimension(40, 30));  
 delFile.setToolTipText("Удалить строку из вкладки");  
  
 Trash = new JButton(new ImageIcon("./img/Trash.png"));  
 Trash.setPreferredSize(new Dimension(40, 30));  
 Trash.setToolTipText("Удаление всей вкладки");  
  
 // Новая кнопка для регистрации пользователя  
 register = new JButton(new ImageIcon("./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);   
  
 // 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", "Начало"};  
 modelSorev = new DefaultTableModel(new String[][]{}, columnsSorev);  
 tableSorev = new JTable(modelSorev);  
 scrollSorev = new JScrollPane(tableSorev);  
  
 JTabbedPane tabbedPane = new JTabbedPane();  
 tabbedPane.addTab("Команды", scrollTeams);  
 tabbedPane.addTab("Гонщики", scrollDrivers);  
 tabbedPane.addTab("Трассы", scrollTracks);  
 tabbedPane.addTab("Соревнования", scrollSorev);  
  
 bookList.add(tabbedPane, BorderLayout.CENTER);  
  
 // Панель поиска  
 searchCriteria = new JComboBox<>(new String[]{"Команды", "Гонщики", "Трассы", "Соревнования"});  
 searchField = new JTextField("Ключевое слово", 30);  
 filter = new JButton("Поиск");  
 resetButton = new JButton("Сбросить");  
 JPanel searchPanel = new JPanel();  
 searchPanel.add(searchCriteria);  
 searchPanel.add(searchField);  
 searchPanel.add(filter);  
 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  
 }  
 });  
   
 // Добавление данных в библиотеку  
   
 addFile.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent arg0) {  
 FileDialog addFileDialog = new FileDialog(bookList, "Загрузка данных из XML", FileDialog.LOAD);  
 addFileDialog.setVisible(true);  
   
 String directory = addFileDialog.getDirectory();  
 String selectedFile = addFileDialog.getFile();  
   
 if (directory == null || selectedFile == null) {  
 return; // Если пользователь нажал «отмена», прекратить выполнение  
 }  
   
 String fileName = directory + selectedFile;  
  
 // Загрузка данных из XML  
 try {  
 File xmlFile = new File(fileName);  
 DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();  
 DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();  
 Document doc = dBuilder.parse(xmlFile);  
 doc.getDocumentElement().normalize();  
  
 // Чтение данных для каждой таблицы  
 loadTableData(doc, "teams", modelTeams);  
 loadTableData(doc, "drivers", modelDrivers);  
 loadTableData(doc, "tracks", modelTracks);  
 loadTableData(doc, "sorev", modelSorev);  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 });  
   
   
   
   
   
 // Удаление данных вкладки  
 Trash.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent event) {  
 // Окно выбора вкладки для удаления данных  
 String[] options = {"Команды", "Гонщики", "Трассы", "Соревнования"};  
 int tabChoice = JOptionPane.showOptionDialog(  
 bookList,   
 "Выберите вкладку, данные которой нужно удалить:",  
 "Выбор вкладки для удаления",  
 JOptionPane.DEFAULT\_OPTION,   
 JOptionPane.QUESTION\_MESSAGE,   
 null,   
 options,   
 options[0]  
 );  
 // Определяем выбранную таблицу  
 DefaultTableModel selectedModel;  
 switch (tabChoice) {  
 case 0: selectedModel = modelTeams; break;  
 case 1: selectedModel = modelDrivers; break;  
 case 2: selectedModel = modelTracks; break;  
 case 3: selectedModel = modelSorev; break;  
 default: return;  
 }  
 int rows = selectedModel.getRowCount();  
 for (int i = 0; i < rows; i++) selectedModel.removeRow(0); // Очистка таблицы  
 }  
 });  
  
 // Сохранение файла  
 save.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent arg0) {  
 FileDialog saveDialog = new FileDialog(bookList, "Сохранение данных в XML", FileDialog.SAVE);  
 saveDialog.setFile("data.xml");  
 saveDialog.setVisible(true);  
  
 String directory = saveDialog.getDirectory();  
 String selectedFile = saveDialog.getFile();  
  
 if (directory == null || selectedFile == null) {  
 return; // Если пользователь нажал «отмена»  
 }  
  
 String fileName = directory + selectedFile;  
  
 // Сохранение данных в XML  
 try {  
 DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();  
 DocumentBuilder docBuilder = docFactory.newDocumentBuilder();  
 Document doc = docBuilder.newDocument();  
  
 // Корневой элемент  
 Element rootElement = doc.createElement("data");  
 doc.appendChild(rootElement);  
  
 // Сохранение данных для каждой таблицы  
 saveTableData(doc, rootElement, "teams", modelTeams);  
 saveTableData(doc, rootElement, "drivers", modelDrivers);  
 saveTableData(doc, rootElement, "tracks", modelTracks);  
 saveTableData(doc, rootElement, "sorev", modelSorev);  
  
 // Запись в файл  
 TransformerFactory transformerFactory = TransformerFactory.newInstance();  
 Transformer transformer = transformerFactory.newTransformer();  
 DOMSource source = new DOMSource(doc);  
 StreamResult result = new StreamResult(new File(fileName));  
 transformer.transform(source, result);  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
  
 // Удаление строки из вкладки  
 delFile.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent event) {  
 // Получение активной вкладки  
 int selectedTab = tabbedPane.getSelectedIndex();  
  
 JTable selectedTable;  
 DefaultTableModel selectedModel;  
  
 // Определяем, какая таблица активна  
 switch (selectedTab) {  
 case 0:  
 selectedTable = tableTeams;  
 selectedModel = modelTeams;  
 break;  
 case 1:  
 selectedTable = tableDrivers;  
 selectedModel = modelDrivers;  
 break;  
 case 2:  
 selectedTable = tableTracks;  
 selectedModel = modelTracks;  
 break;  
 case 3:  
 selectedTable = tableSorev;  
 selectedModel = modelSorev;  
 break;  
 default:  
 return;  
 }  
  
 // Получаем индекс выбранной строки в активной таблице  
 int selectedRow = selectedTable.getSelectedRow();  
  
 // Проверяем, выбрана ли строка  
 if (selectedRow != -1) {  
 selectedModel.removeRow(selectedRow); // Удаляем строку  
 } 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);  
 }  
   
 //////////////////////////////////////////////////////METHODS//////////////////////////////////////////////////////\  
 */\*\*  
 \* Извлекает данные из переданной модели 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 для всех вкладок.  
 \* Применяется для таблиц: команды, гонщики, трассы и соревнования.  
 \*/* private void enableSorting() {  
 tableTeams.setAutoCreateRowSorter(true);  
 tableDrivers.setAutoCreateRowSorter(true);  
 tableTracks.setAutoCreateRowSorter(true);  
 tableSorev.setAutoCreateRowSorter(true);  
 }  
   
 */\*\*  
 \* загружает ХМЛ файлы  
 \*/* private void loadTableData(Document doc, String tagName, DefaultTableModel model) {  
 NodeList nodeList = doc.getElementsByTagName(tagName); // Получаем все элементы с заданным тегом  
 if (nodeList.getLength() > 0) {  
 NodeList rowList = nodeList.item(0).getChildNodes(); // Получаем строки внутри элемента  
  
 for (int i = 0; i < rowList.getLength(); i++) {  
 Node rowNode = rowList.item(i);  
 if (rowNode.getNodeType() == Node.ELEMENT\_NODE) {  
 Element rowElement = (Element) rowNode;  
 String[] rowData = new String[model.getColumnCount()]; // Создаем массив для данных строки  
  
 // Проходим по всем столбцам в строке  
 for (int j = 0; j < model.getColumnCount(); j++) {  
 String columnName = model.getColumnName(j);  
 NodeList cellNodes = rowElement.getElementsByTagName(columnName);  
 if (cellNodes.getLength() > 0) {  
 rowData[j] = cellNodes.item(0).getTextContent(); // Получаем текстовое содержимое ячейки  
 } else {  
 rowData[j] = ""; // Если ячейка не найдена, добавляем пустую строку  
 }  
 }  
 model.addRow(rowData); // Добавляем строку в модель таблицы  
 }  
 }  
 }  
 }  
  
  
   
 */\*\*  
 \* сохраняет ХМЛ файлы  
 \*/* private void saveTableData(Document doc, Element rootElement, String tagName, DefaultTableModel model) {  
 Element tableElement = doc.createElement(tagName);  
 rootElement.appendChild(tableElement);  
  
 // Проходим по всем строкам таблицы  
 for (int i = 0; i < model.getRowCount(); i++) {  
 Element rowElement = doc.createElement("row");  
 tableElement.appendChild(rowElement);  
  
 // Проходим по всем столбцам в строке  
 for (int j = 0; j < model.getColumnCount(); j++) {  
 Element cellElement = doc.createElement(model.getColumnName(j)); // Имя элемента соответствует заголовку столбца  
 cellElement.appendChild(doc.createTextNode((String) model.getValueAt(i, j))); // Добавляем текстовое содержимое  
 rowElement.appendChild(cellElement); // Добавляем ячейку в строку  
 }  
 }  
 }  
  
  
 */\*\*  
 \* Сбрасывает поиск и восстанавливает исходные данные для выбранной вкладки.  
 \* Используется для восстановления таблиц после фильтрации.  
 \*/* 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 (вкладке).  
 \* Фильтрует строки, которые содержат введенное ключевое слово.  
 \*/* 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.  
 \* Поля ввода включают имя пользователя и пароль.   
 \* Реализована проверка корректности ввода.  
 \*/* 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  
 }  
 });  
  
 // 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, который служит точкой входа в приложение.  
 \* Запускает окно приложения {@code BookList} и показывает его пользователю.  
 \*  
 \* @param args аргументы командной строки, передаваемые в программу (не используются).  
 \*/* public static void main(String[] args) {  
 // Запуск основного окна приложения  
 new BookList().show();  
 }  
}