Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический университет

“ЛЭТИ” им.В.И.Ульянова (Ленина) »

Кафедра ВТ

ОТЧЕТ
по лабораторно-практической работе № 4
«Обработка исключений»

Выполнил Зайцев И.С.

Факультет КТИ

Группа № 3312

Подпись преподавателя _____

Санкт-Петербург

2024 г

Цель работы

Знакомство с механизмом обработки исключений в языке.

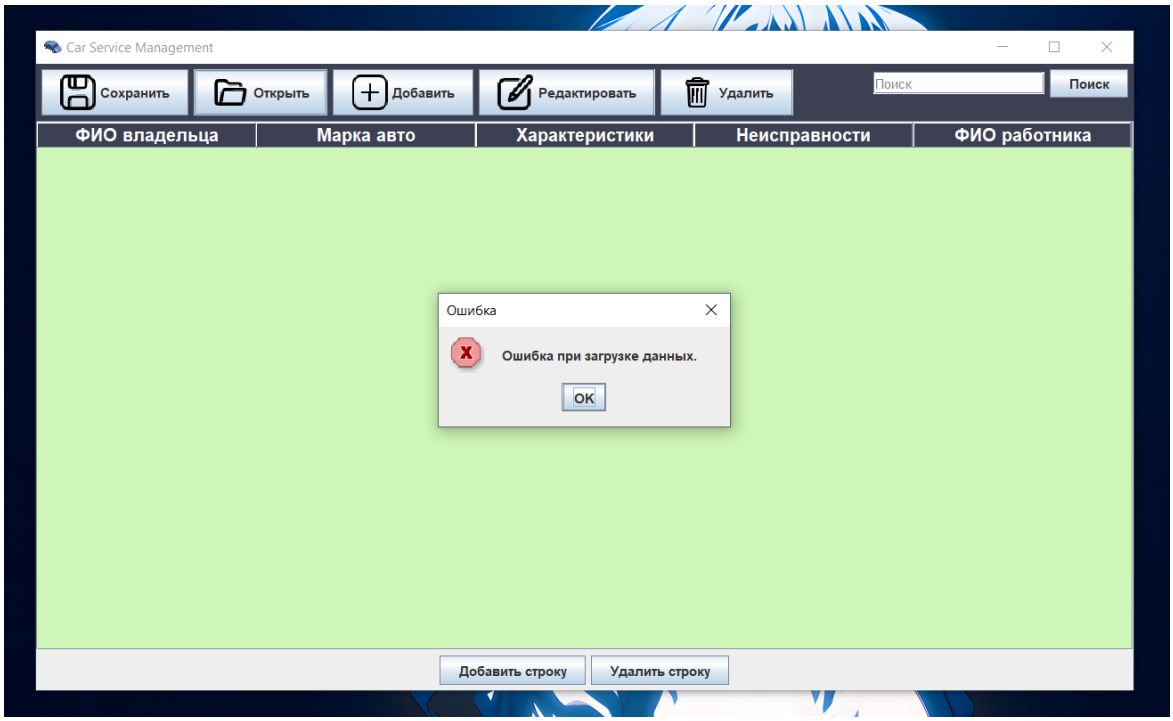
Перечень ситуаций, которые контролируются с помощью исключений.

- **Ошибка конфигурации парсера XML**:
  - **Метод**: `Save_To_XML`
  - **Ситуация**: Ошибка возникает при создании экземпляра `DocumentBuilder`. В случае возникновения `ParserConfigurationException` выбрасывается пользовательское исключение `MyException` с сообщением "Ошибка при сохранении данных."
- **Ошибка сохранения XML файла**:
  - **Метод**: `Save_To_XML`
  - **Ситуация**: Ошибка происходит при трансформации документа в поток результата (файл). При возникновении `TransformerException` выбрасывается `MyException` с сообщением "Ошибка при сохранении данных."
- **Ошибка при загрузке XML файла**:
  - **Метод**: `Open_XML`
  - **Ситуация**: Ошибка может возникнуть при парсинге XML документа, что может привести к `ParserConfigurationException`, `IOException`, или `SAXException`. В этом случае выбрасывается `MyException` с сообщением "Ошибка при загрузке данных."
- **Удаление строки из таблицы**:
  - **Метод**: `deleteRowButton` ActionListener

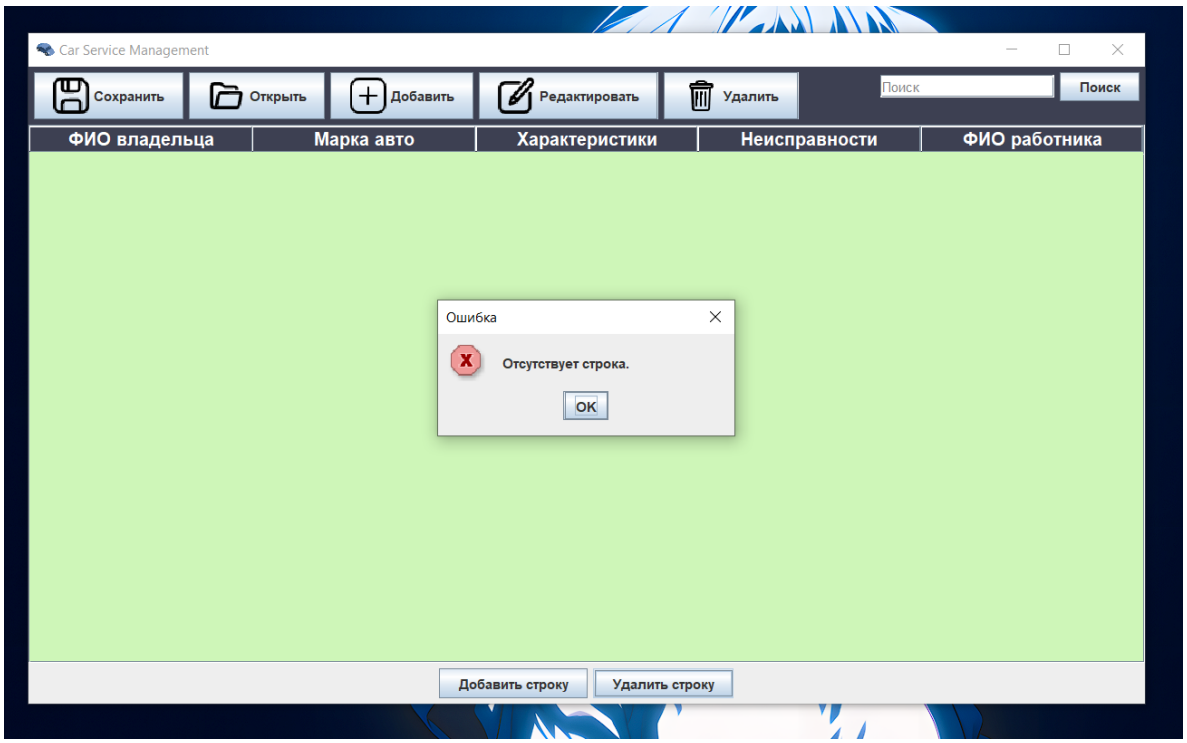
- **Ситуация**: Если попытаться удалить строку из пустой таблицы, показывается диалоговое окно с сообщением об ошибке "Отсутствует строка."

Скриншоты, иллюстрирующие работу слушателей

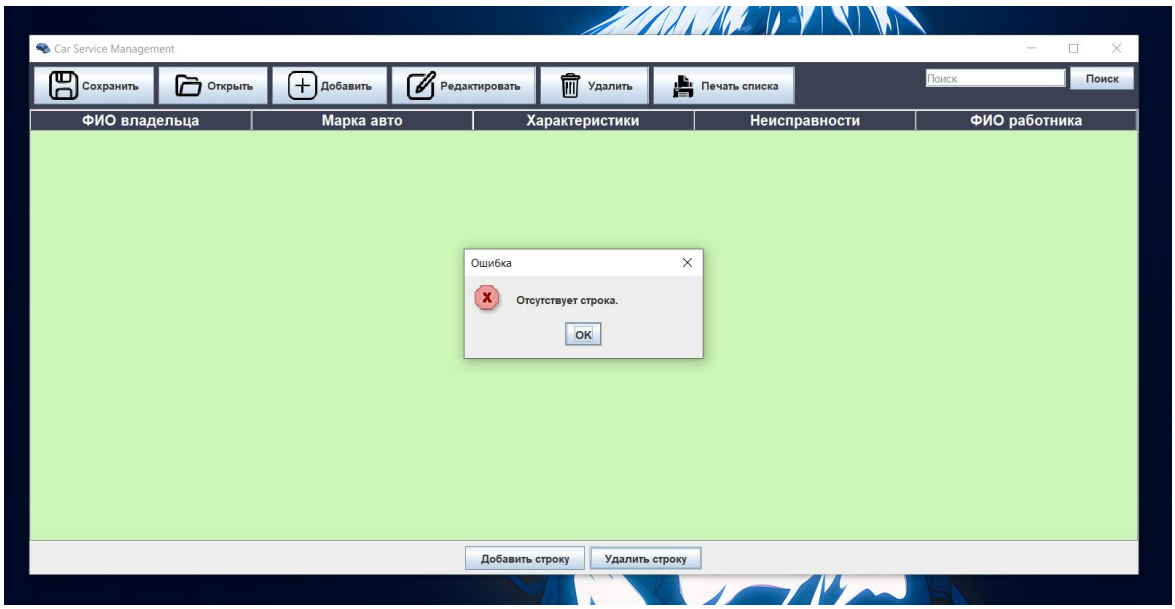
Пример 1



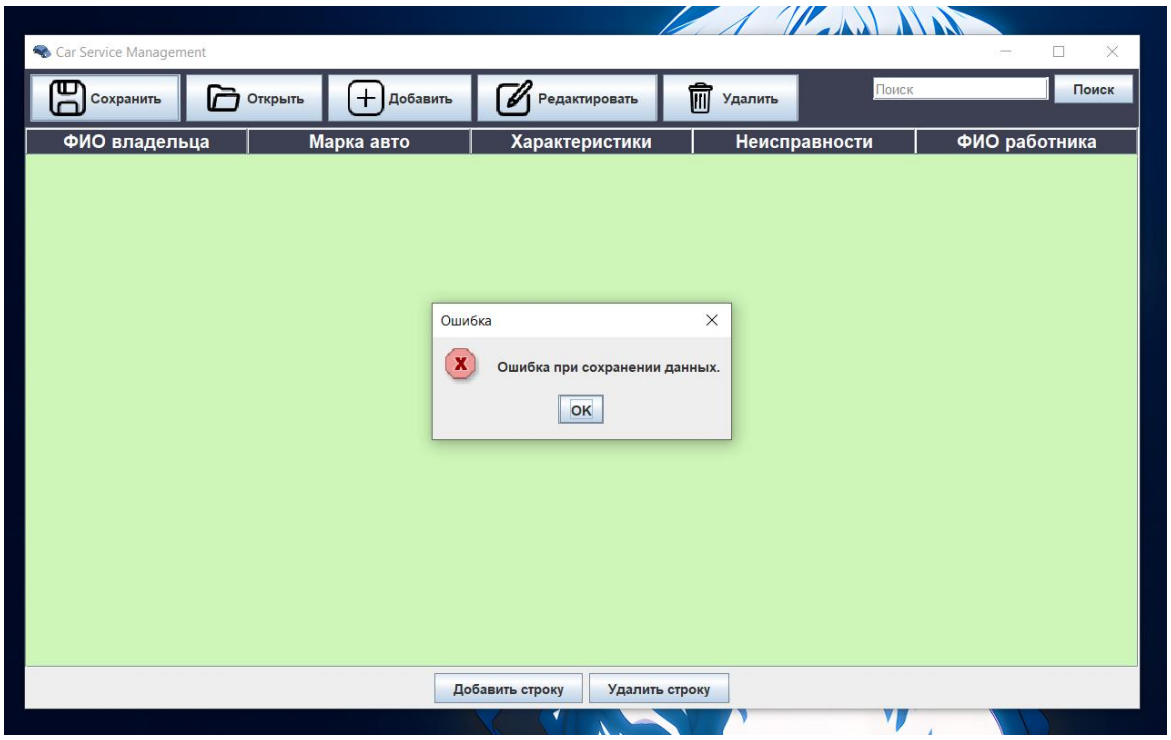
Пример 2



Пример 3



Пример 4



Текст программы

Класс App:

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableModel;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusAdapter;
import java.awt.event.FocusEvent;
import java.io.File;
import java.io.IOException;
import java.util.Objects;

public class App {
    /**
     * Метод инициализирует и отображает окно приложения для управления
автосервисом.
     * Создает основное окно с панелями, кнопками и таблицей для отображения
информации.
     */
    public static void CarService() {
        // Создание главного окна приложения
```

```
        JFrame frame = new JFrame("Car Service Management");
        // Установка иконки для окна
        ImageIcon icon = new
ImageIcon(Objects.requireNonNull(App.class.getResource("icons\\car.png")));
        frame.setIconImage(icon.getImage());
        // Настройка действия при закрытии окна
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 600); // Установка размера окна
        // Установка цвета фона окна
        frame.getContentPane().setBackground(new Color(61, 64, 82));

        // Главная панель, которая содержит кнопки и панель поиска
        JPanel topPanel = new JPanel();
        topPanel.setLayout(new BorderLayout()); // Используем BorderLayout
для организации кнопок и поиска
        // Панель для кнопок
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
        buttonPanel.setBackground(new Color(61, 64, 82));

        // Массивы с названиями и иконками для кнопок
        String[] icons = {"save", "open", "add", "edit", "bin", "print"};
        String[] buttonsName = {"Сохранить", "Открыть", "Добавить",
"Редактировать", "Удалить", "Печать списка"};
        JButton[] buttons = new JButton[icons.length];

        // Добавляем кнопки на панель кнопок
        for (int i = 0; i < icons.length; i++) {
            ImageIcon iconImage = new ImageIcon(new

ImageIcon(Objects.requireNonNull(App.class.getResource("icons\\" + icons[i] +
".png")))
                    .getImage().getScaledInstance(32, 32,
java.awt.Image.SCALE_SMOOTH));
            buttons[i] = new JButton(buttonsName[i], iconImage);
            buttonPanel.add(buttons[i]);
        }

        for (int i = 2; i < buttons.length; i++) {
            buttons[i].addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    // При нажатии на кнопку открываем диалоговое окно с
сообщением
                    JOptionPane.showMessageDialog(frame, "Пока в работе",
"Информация", JOptionPane.INFORMATION_MESSAGE);
                }
            });
        }

        // Создание панели для поиска
        JPanel searchPanel = new JPanel();
        searchPanel.setLayout(new FlowLayout(FlowLayout.RIGHT)); //
Выравниваем по правому краю
        searchPanel.setBackground(new Color(61, 64, 82));
        // Поле для ввода текста поиска
        JTextField searchField = new JTextField(15);
        JButton searchButton = new JButton("Поиск"); // Кнопка для поиска
        // Добавляем текст плейсхолдера в поле поиска
        searchField.setText("Поиск");
        searchField.setForeground(Color.GRAY); // По умолчанию серый текст
        // Добавляем FocusListener для обработки фокуса поля

        /**
```

```
         * Добавляет обработчик событий фокуса для текстового поля поиска.
         * Когда фокус получен, плейсхолдер удаляется, и текст становится
         черным.
         * Когда фокус потерян, если поле пустое, плейсхолдер возвращается.
         *
         * @param FocusAdapter - адаптерный класс, который предоставляет
        пустую реализацию методов интерфейса FocusListener.
         * Данный интерфейс включает методы focusGained() и focusLost().
         */
        searchField.addFocusListener(new FocusAdapter() {
            /**
             * Метод вызывается, когда фокус на текстовое поле поиска
             получен.
             * Если текст равен "Поиск" (плейсхолдер), он удаляется, и цвет
             текста меняется на черный.
             *
             * @param e - событие фокуса.
             */
            @Override
            public void focusGained(FocusEvent e) {
                if (searchField.getText().equals("Поиск")) {
                    searchField.setText("");
                    searchField.setForeground(Color.BLACK); // Цвет текста
черный при вводе
                }
            }

            /**
             * Метод вызывается, когда текстовое поле поиска теряет фокус.
             * Если поле пустое, возвращается текст плейсхолдера "Поиск", и
             цвет текста меняется на серый.
             *
             * @param e - событие фокуса.
             */
            @Override
            public void focusLost(FocusEvent e) {
                if (searchField.getText().isEmpty()) {
                    searchField.setForeground(Color.GRAY);
                    searchField.setText("Поиск"); // Возвращаем плейсхолдер
                }
            }
        });

        // Добавляем поле поиска и кнопку в панель поиска
        searchPanel.add(searchField);
        searchPanel.add(searchButton);

        // Добавляем панели кнопок и поиска в верхнюю панель
        topPanel.add(buttonPanel, BorderLayout.CENTER);
        topPanel.add(searchPanel, BorderLayout.EAST); // Панель поиска справа

        // Заголовки столбцов таблицы
        String[] columnNames = {"ФИО владельца", "Марка авто",
"Характеристики", "Неисправности", "ФИО работника"};

        // Модель таблицы с возможностью добавления/удаления строк
        DefaultTableModel model = new DefaultTableModel(columnNames, 0);
        JTable table = new JTable(model); // Таблица с динамической моделью
        table.setBackground(new Color(206, 246, 184));
        table.setFillsViewportHeight(true); // Растягиваем таблицу на всю
доступную высоту
        table.setFont(new Font("Arial", Font.PLAIN, 14));

        // Настраиваем заголовок таблицы
```

```
        JTableHeader header = table.getTableHeader();
        header.setBackground(new Color(61, 64, 82));
        header.setForeground(Color.WHITE);
        header.setFont(new Font("Arial", Font.BOLD, 16));

        // Добавляем таблицу в JScrollPane для прокрутки
        JScrollPane scrollPane = new JScrollPane(table);

        // Панель для добавления/удаления строк
        JPanel actionPanel = new JPanel();
        JButton addRowButton = new JButton("Добавить строку");
        JButton deleteRowButton = new JButton("Удалить строку");

        // Обработчик добавления строки
        /**
         * Добавляет пустую строку в таблицу.
         *
         * @param e событие ActionEvent, возникающее при нажатии кнопки
добавления строки.
         */
        addRowButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Добавляем пустую строку
                model.addRow(new Object[]{"", "", "", "", ""});
            }
        });

        // Обработчик удаления строки
        /**
         * Удаляет последнюю строку из таблицы, если она существует.
         * Если строк нет, выводит сообщение об ошибке.
         *
         * @param e событие ActionEvent, возникающее при нажатии кнопки
удаления строки.
         */
        deleteRowButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int lastRow = model.getRowCount() - 1; // Получаем индекс
последней строки
                if (lastRow >= 0) { // Проверяем, что строка существует
                    model.removeRow(lastRow); // Удаляем последнюю строку
                } else {
                    JOptionPane.showMessageDialog(frame, "Отсутствует
строка.", "Ошибка", JOptionPane.ERROR_MESSAGE);
                }
            }
        });


        // Обработчик для кнопки "Сохранить"
        /**
         *
         * Сохранение таблицы в формате xml файла
         * @param e- событие ActionListener, возникающее при нажатии кнопки
сохранения таблицы.
         */
        buttons[0].addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    Save_To_XML(table, frame);
                } catch (MyException ex) {
```

```
                    JOptionPane.showMessageDialog(frame, ex.getMessage(),
"Ошибка", JOptionPane.ERROR_MESSAGE);
                }
            }
        });

        /**
         *
         * Обработка второй кнопки: открытие (парсинг) xml файла
         * @param e- событие ActionListener, возникающее при нажатии кнопки
открытия таблицы.
         */

        buttons[1].addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    Open_XML(model, frame);
                } catch (MyException ex) {
                    JOptionPane.showMessageDialog(frame, ex.getMessage(),
"Ошибка", JOptionPane.ERROR_MESSAGE);
                }
            }
        });

        // Добавляем кнопки на панель действий
        actionPanel.add(addRowButton);
        actionPanel.add(deleteRowButton);

        // Добавляем верхнюю панель, таблицу и панель действий в окно
        frame.add(topPanel, BorderLayout.NORTH); // Верхняя панель
        frame.add(scrollPane, BorderLayout.CENTER); // Таблица в центре
        frame.add(actionPanel, BorderLayout.SOUTH); // Панель действий снизу

        frame.setVisible(true);
    }

    /**
     *
     * @param table - таблица, передаваемая в метод, для получения из нее
данных
     * @param frame - окно
     * @throws MyException - возникает при ошибке создания XML файла,
выбрасывается при возникновении ParserConfigurationException
     */

    private static void Save_To_XML(JTable table, Frame frame) throws
MyException {
        // Получаем модель данных из таблицы
        TableModel data = table.getModel();
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder;
        Document doc;
        String message = "Ошибка при сохранении данных.";

        // Инициализация Document для создания XML документа
        try {
            builder = factory.newDocumentBuilder();
            doc = builder.newDocument();
        } catch (ParserConfigurationException e) {
            // Генерация пользовательского исключения при ошибке
            throw new MyException(message);
        }
```

```
        // Создаем корневой элемент <carService>
        Element rootElement = doc.createElement("carService");
        doc.appendChild(rootElement);

        // Цикл по строкам таблицы для сохранения данных в XML
        for (int i = 0; i < data.getRowCount(); i++) {
            // Создаем элемент <car> для каждой записи
            Element car = doc.createElement("car");
            rootElement.appendChild(car);

            // Заполняем XML данными из таблицы
            Element owner = doc.createElement("owner");
            owner.appendChild(doc.createTextNode(data.getValueAt(i,
0).toString()));
            car.appendChild(owner);

            Element brand = doc.createElement("brand");
            brand.appendChild(doc.createTextNode(data.getValueAt(i,
1).toString()));
            car.appendChild(brand);

            Element characteristics = doc.createElement("characteristics");
            characteristics.appendChild(doc.createTextNode(data.getValueAt(i,
2).toString()));
            car.appendChild(characteristics);

            Element issues = doc.createElement("issues");
            issues.appendChild(doc.createTextNode(data.getValueAt(i,
3).toString()));
            car.appendChild(issues);

            Element worker = doc.createElement("worker");
            worker.appendChild(doc.createTextNode(data.getValueAt(i,
4).toString()));
            car.appendChild(worker);
        }

        // Сохранение документа XML в файл
        TransformerFactory transformerFactory =
TransformerFactory.newInstance();
        Transformer transformer;
        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(new
File("path\\car_service_data.xml"));

        /**
         * @param e- исключение, возникающее при ошибке сохранения XML файла
         */
        try {
            transformer = transformerFactory.newTransformer();
            transformer.transform(source, result);
        } catch (TransformerException e) {
            // Обработка исключения при ошибке сохранения
            throw new MyException(message);
        }

        // Уведомление пользователя об успешном сохранении
        JOptionPane.showMessageDialog(frame, "Данные успешно сохранены в XML
файл.", "Успех", JOptionPane.INFORMATION_MESSAGE);
    }

    /**
     *
```

```
     * @param model - Динамическая таблица
     * @param frame - окно
     * @throws MyException - возникает при ошибке парсинга XML файла,
выбрасывается при возникновении ParserConfigurationException | IOException |
SAXException
     */

    private static void Open_XML(DefaultTableModel model, JFrame frame)
throws MyException {
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder;
        Document doc;

        // Инициализация и загрузка XML документа
        try {
            builder = factory.newDocumentBuilder();
            doc = builder.parse(new File("path\\car_service_data.xml"));
        } catch (ParserConfigurationException | IOException | SAXException e)
{
            // Обработка ошибок при загрузке XML файла
            throw new MyException("Ошибка при загрузке данных.");
        }

        // Нормализуем структуру XML (удаляем излишние пробелы и пустые
элементы)
        doc.getDocumentElement().normalize();

        // Получаем корневой элемент <carService>
        Element root = doc.getDocumentElement();

        // Очищаем текущие данные таблицы перед загрузкой новых
        model.setRowCount(0);

        // Получаем список всех элементов <car>
        NodeList carList = root.getElementsByTagName("car");

        // Цикл по элементам <car> для загрузки данных в таблицу
        for (int i = 0; i < carList.getLength(); i++) {
            Node carNode = carList.item(i);

            if (carNode.getNodeType() == Node.ELEMENT_NODE) {
                Element carElement = (Element) carNode;

                // Извлекаем данные владельца, марки авто и других
характеристик
                String owner =
carElement.getElementsByTagName("owner").item(0).getTextContent();
                String brand =
carElement.getElementsByTagName("brand").item(0).getTextContent();
                String characteristics =
carElement.getElementsByTagName("characteristics").item(0).getTextContent();
                String issues =
carElement.getElementsByTagName("issues").item(0).getTextContent();
                String worker =
carElement.getElementsByTagName("worker").item(0).getTextContent();

                // Добавляем строку с загруженными данными в таблицу
                model.addRow(new Object[]{owner, brand, characteristics,
issues, worker});
            }
        }

        // Уведомление пользователя об успешной загрузке
```

```
        JOptionPane.showMessageDialog(frame, "Данные успешно распакованы.",
"Успех", JOptionPane.INFORMATION_MESSAGE);

        frame.setVisible(true);
    }

    // Класс для обработки ошибок с пользовательскими сообщениями

    public static class MyException extends Exception {
        /**
         *
         * @param message - String значения для обозначения exception
         */
        public MyException(String message) {
            super(message);
        }
    }
}
```

Класс Main:

```
/**
 * Лабораторная работа №3.
 *
 * @author Илья Зайцев 3312;
 * @version 1.0;
 *
 */
public class Main {
  /**
   * @param args - вводимая строка (параметр запуска);
   */
  public static void main(String[] args) {
    App.CarService();
  }
}
```

Дополнение:

В этом коде я добавил свой класс исключений `MyException`, чтобы упростить обработку ошибок при работе с XML-файлами. Теперь методы `Save_To_XML` и `Open_XML` выбрасывают это исключение при возникновении ошибок, таких как проблемы с конфигурацией парсера или сохранением данных. Такой подход позволил централизовать и стандартизировать обработку исключений, что облегчает отладку и делает код более структурированным.

Выводы

В этом проекте я добавил кастомный класс исключений `MyException` для обработки ошибок, связанных с сохранением и загрузкой данных в XML. Этот класс

позволяет задавать конкретные сообщения об ошибках, что делает процесс отладки и информирования пользователя более понятным. Методы, работающие с XML-файлами, теперь пробрасывают `MyException` при возникновении ошибок, что позволяет мне более гибко управлять ошибками и отображать информативные сообщения для пользователя.

Ссылка на репозиторий с видео и JavDoc файлом:

https://github.com/IlyaZaytsev26/OOP_Java