

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 № 5
«Сохранение и загрузка данных из файла»

Выполнил Зайцев И.С.

Факультет КТИ

Группа № 3312

Подпись преподавателя _____

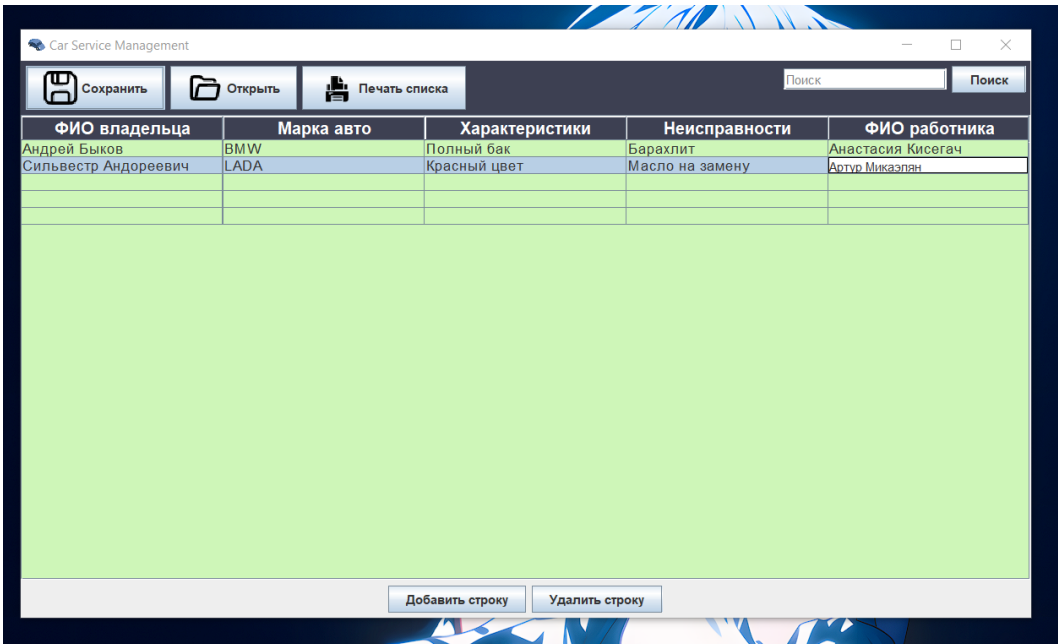
Санкт-Петербург
2024 г

Цель работы

Знакомство с организацией обмена данными между объектами экранной формы и файлом.

Распечатки содержимого файлов с данными до и после внесения изменений.

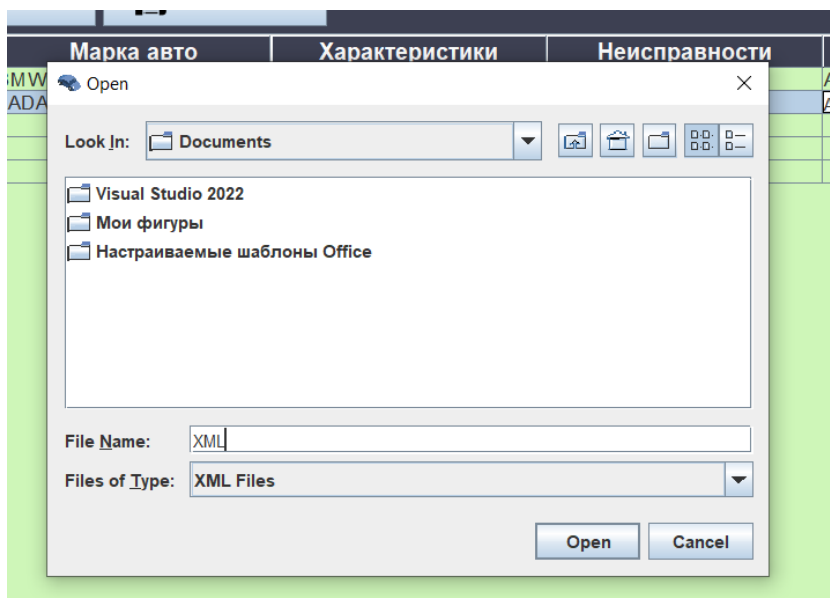
Данные в таблице



The screenshot shows a window titled 'Car Service Management'. It has a menu bar with 'Сохранить' (Save), 'Открыть' (Open), and 'Печать списка' (Print list). There is a search bar with 'Поиск' (Search) on the right. Below the menu is a table with five columns: 'ФИО владельца' (Owner's name), 'Марка авто' (Car brand), 'Характеристики' (Features), 'Неисправности' (Malfunctions), and 'ФИО работника' (Employee's name). The table contains two rows of data. Below the table are two buttons: 'Добавить строку' (Add row) and 'Удалить строку' (Delete row).

ФИО владельца	Марка авто	Характеристики	Неисправности	ФИО работника
Андрей Быков	BMW	Полный бак	Барахлит	Анастасия Кисегач
Сильвестр Андореевич	LADA	Красный цвет	Масло на замену	Артур Микаэлян

Процесс сохранения:



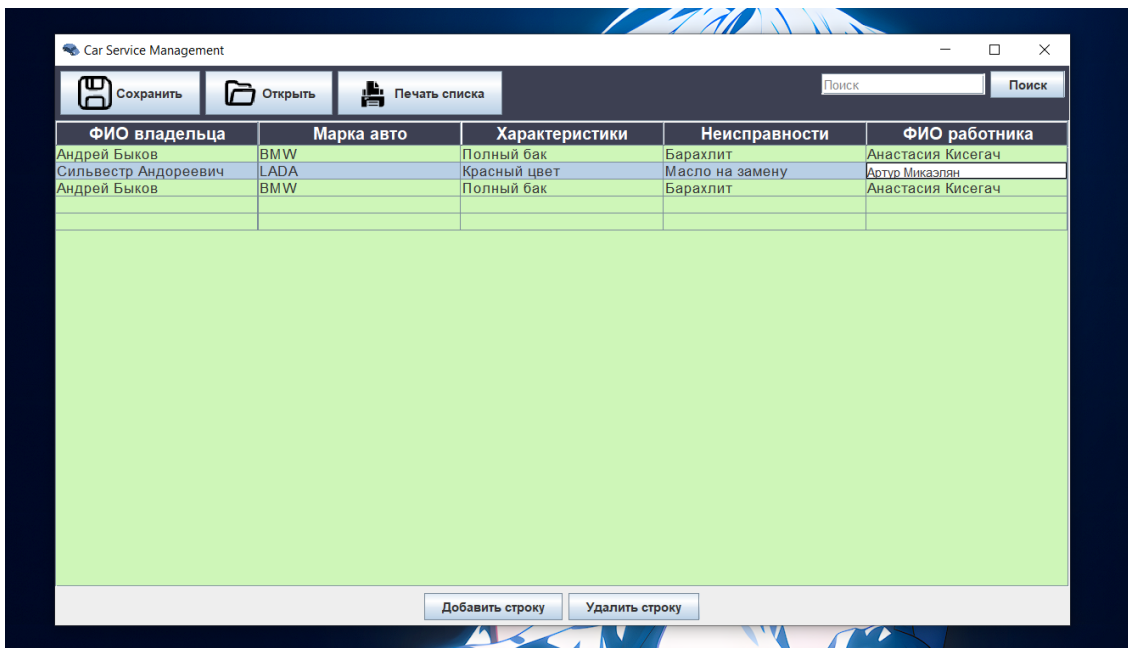
Данные в XML файле:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><carService><car><owner>Андрей Быков</owner><brand>BMW</brand><characteristics>Полный бак</characteristics><issues>Бараклит</issues><worker>Анастасия Кисегач</worker></car><car><owner>Сильвестр Андреевич</owner><brand>LADA</brand><characteristics>Красный цвет</characteristics><issues>Масло на замену</issues><worker></car></car><owner><brand><characteristics><issues><worker></car></carService>
```

Добавим ещё один автомобиль в конец в файл

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><carService><car><owner>Андрей Быков</owner><brand>BMW</brand><characteristics>Полный бак</characteristics><issues>Бараклит</issues><worker>Анастасия Кисегач</worker></car><car><owner>Сильвестр Андреевич</owner><brand>LADA</brand><characteristics>Красный цвет</characteristics><issues>Масло на замену</issues><worker></car></car><car><owner>Сильвестр Андреевич</owner><brand>LADA</brand><characteristics>Красный цвет</characteristics><issues>Масло на замену</issues><worker></car></carService>
```

Откроем файл в программе



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.filechooser.FileNameExtensionFilter;
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 extends Component {
    /**
     * 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", "print"};
        String[] buttonsName = {"Сохранить", "Открыть", "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]);
        }

        buttons[2].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);
        }
    }
});

JFileChooser fileChooser = new JFileChooser();
// 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, Get_Name(fileChooser, 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, Get_Name(fileChooser, 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);
}

private static String Get_Name(JFileChooser fileChooser, JFrame frame){
    fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
    fileChooser.setAcceptAllFileFilterUsed(false);
    FileNameExtensionFilter extFilter = new FileNameExtensionFilter("XML
Files", "xml");
    fileChooser.addChoosableFileFilter(extFilter);
    int name = fileChooser.showOpenDialog(frame);

    return fileChooser.getSelectedFile().toString();
}

/**
 *
 * @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, String
name_to_file) 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(name_to_file));

/**
 * @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,
String name_of_file) throws MyException {
    DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
    DocumentBuilder builder;
    Document doc;

    // Инициализация и загрузка XML документа
    try {
        builder = factory.newDocumentBuilder();
        doc = builder.parse(name_of_file);
    } 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();
    }
}

```

Выводы

В этой работе я добавил функции для работы с XML, чтобы реализовать обмен данными между формой и файлом в приложении для управления автосервисом. Я также создал класс исключений `MyException`, чтобы обрабатывать ошибки при сохранении и загрузке XML, что позволяет отображать более понятные сообщения пользователю и улучшает отладку. Ссылка на репозиторий с видео и Javadoc файлом:

https://github.com/IlyaZaytsev26/OOP_Java