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 МОЭВМ

ОТЧЕТ

по лабораторно-практической работе № 6

«Обработка XML-документов»

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

Выполнил: Локтионов Т. И.

Факультет КТИ

Группа № 3311

Подпись преподавателя _____

Санкт-Петербург

2024 г

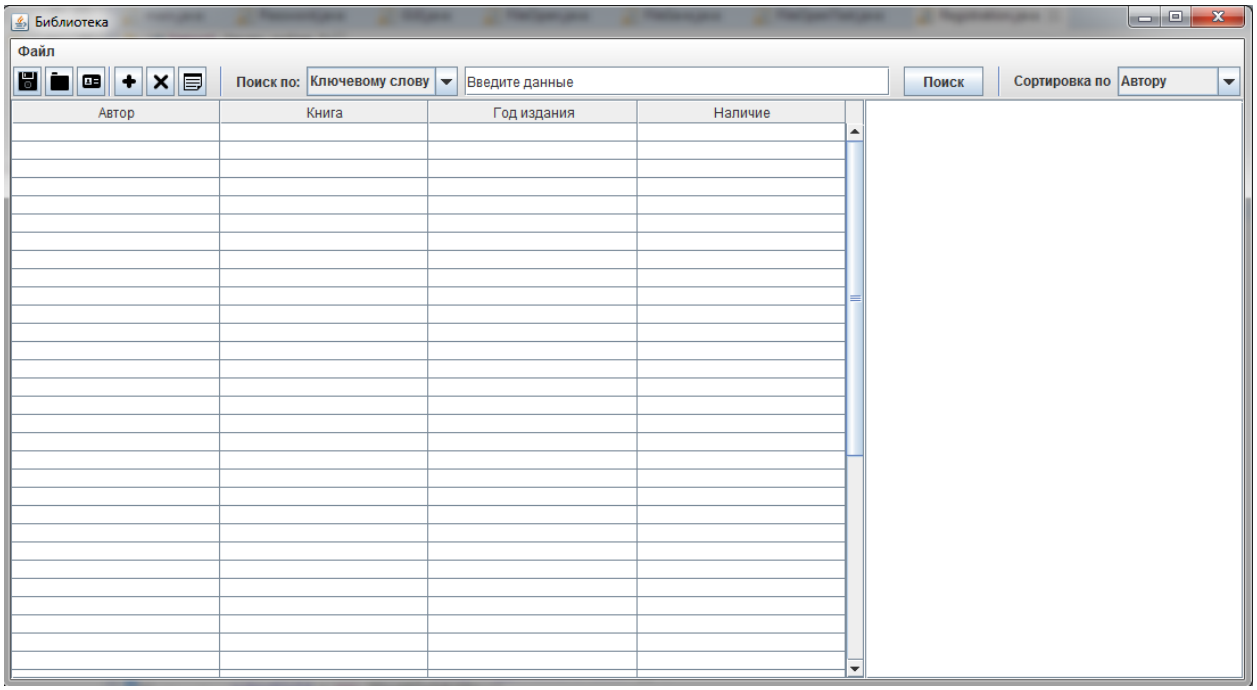
Цель работы:
знакомство с технологией обработки XML-документов и файлов.

Описание задания:
1. Создать XML файл с деревом данных
2. Написать и заменить слушатели для загрузки и выгрузки данных из файла
3. Проверить работоспособность программы

Описание экранной формы:

Экранная форма предназначена для отображения списка больных и врачей для администратора регистратуры поликлиники, она может менять свой размер на экране (начальный размер 800x600). Форма должна реализовывать следующие функции: загрузку списка пациентов, болезней, врачей, дат приема и состояния приема из файла и выгрузку этой информации в файл; редактирование списка, включая: добавление, удаление, корректировку информации; удобный поиск, по ключевым словам, и/или другими методами (имя пациента, дата и т.д.)

Макет (взят из приложенных к методичке файлов):

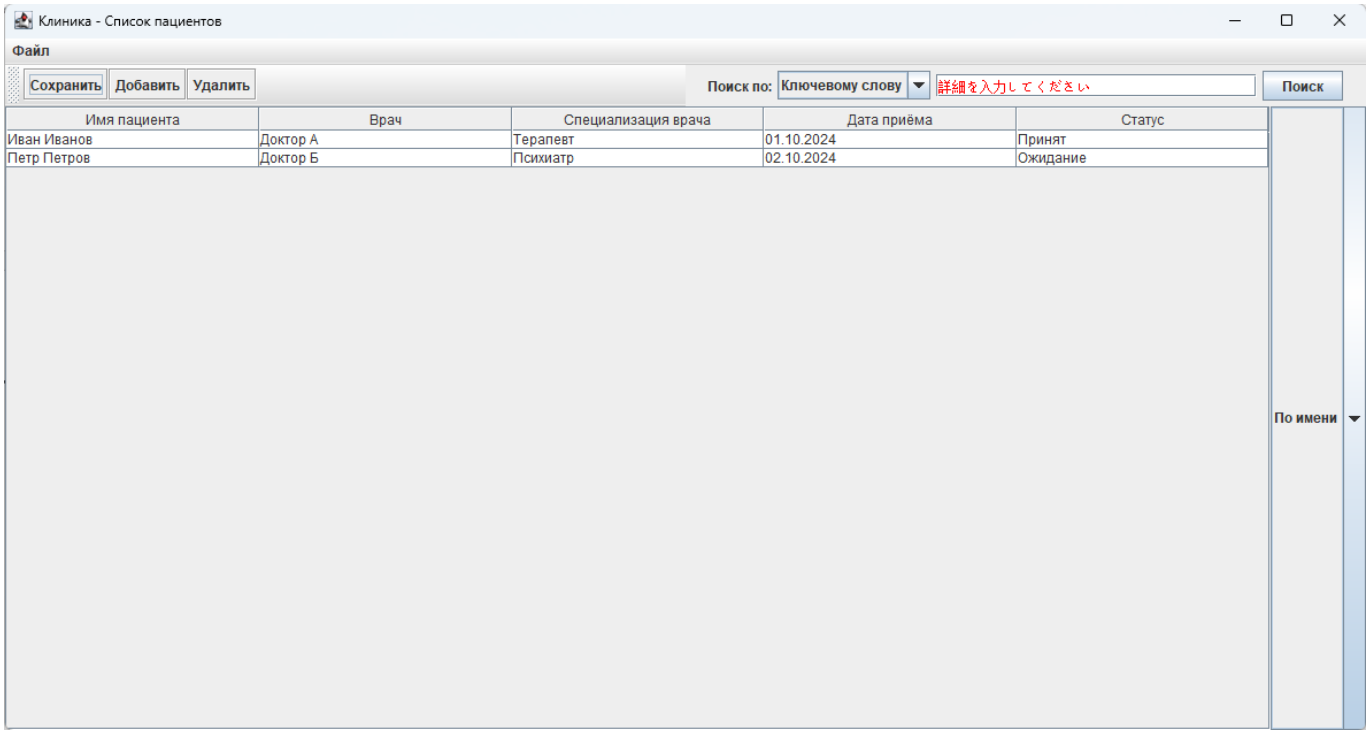


Перечень возможных исключений:

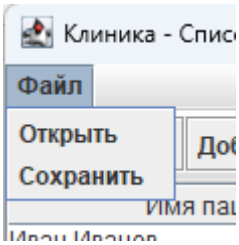
- Пустые поля ввода, которые недопустимы при добавлении пациента в таблицу.
- Некорректные данные, которые не соответствуют установленным требованиям (например, неверный формат).
- Дублирование записей, когда пользователь пытается добавить существующего пациента.
- Неожиданные ошибки, возникающие в ходе выполнения программы, которые могут быть перехвачены и обработаны.

Работоспособность приложения:

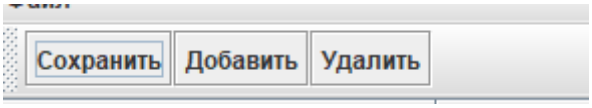
JFrame:



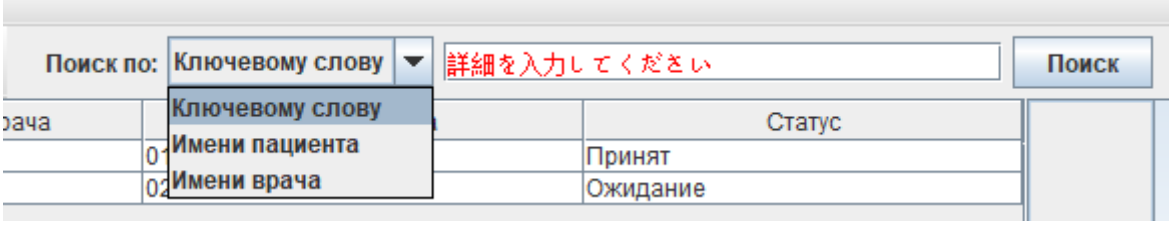
JMenuBar:



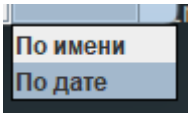
JToolBar:



JPanel && JTextField:

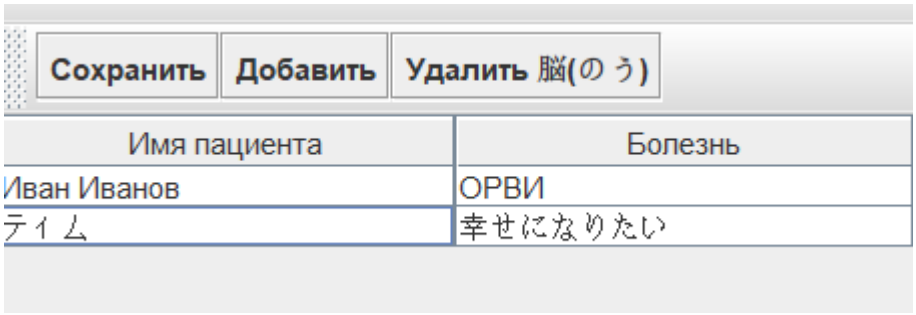


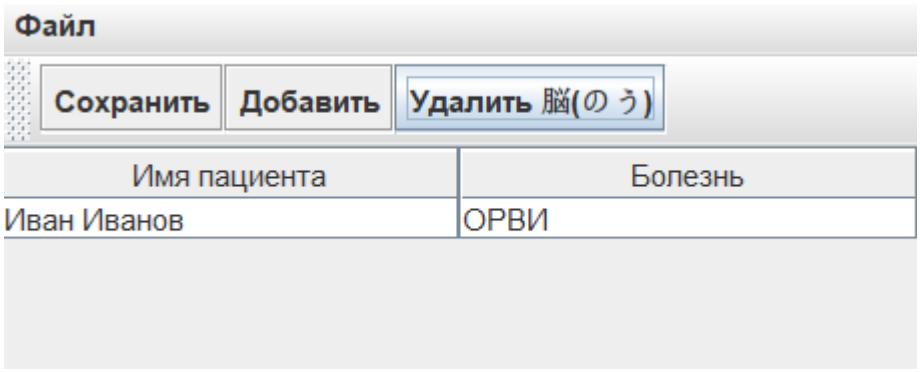
JComboBox:



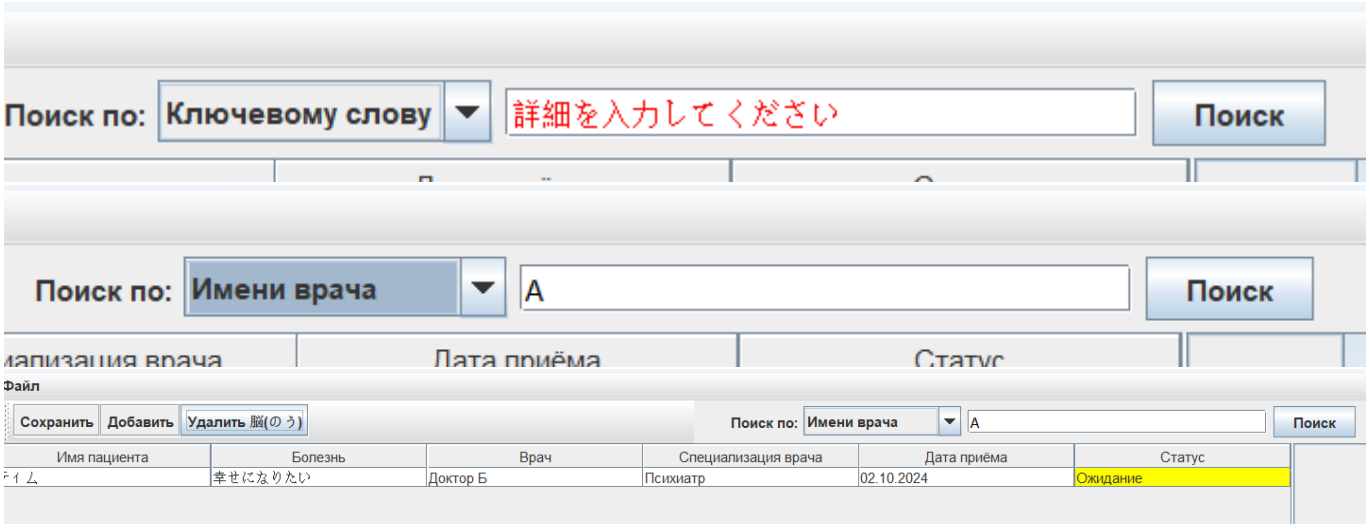
Работоспособность слушателей:

deleteButton:



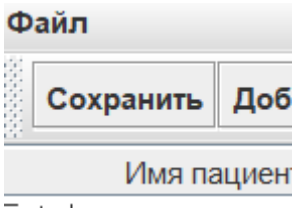


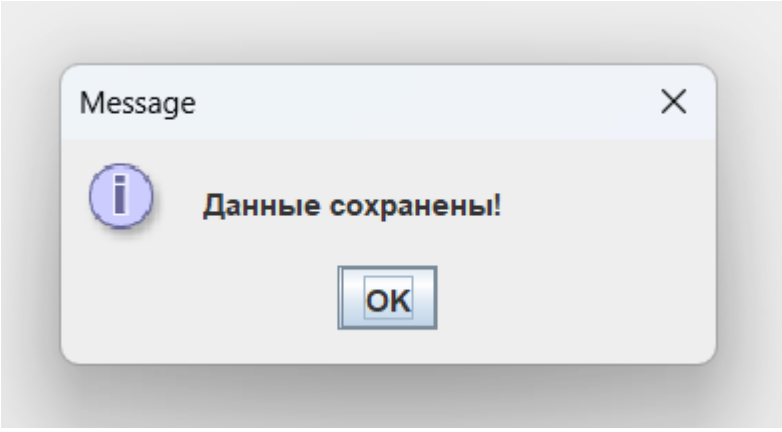
searchField && searchButton:



// работает как поиск по ключевому слову, так и изменение поля поиска при начале ввода

saveButton:

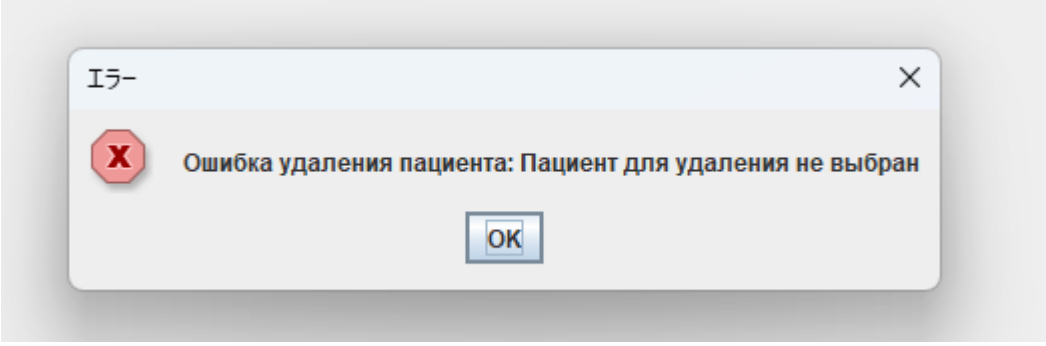




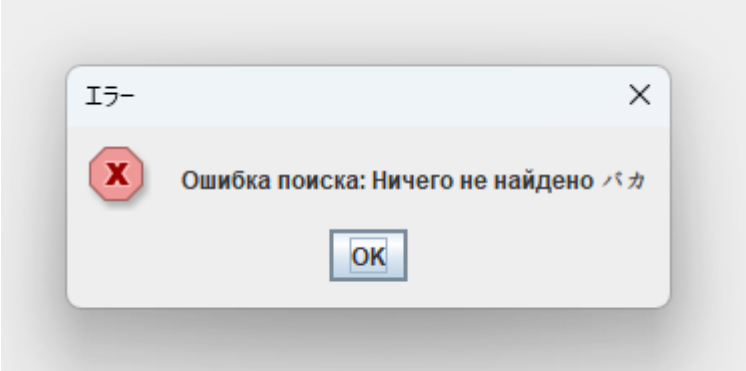
// только вывод сообщения

Работоспособность исключений:

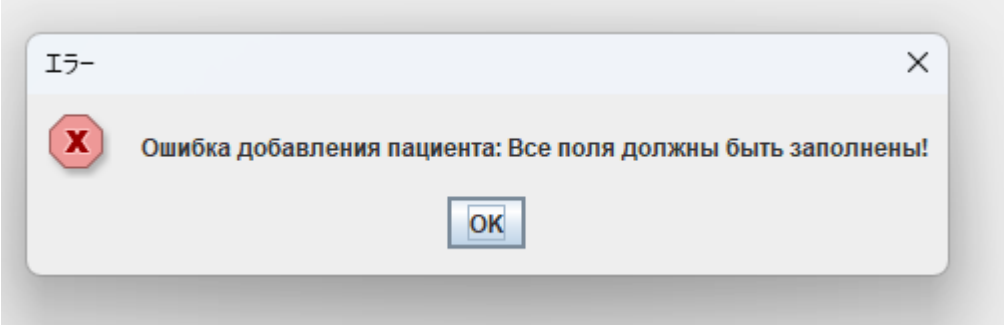
// deleteButton



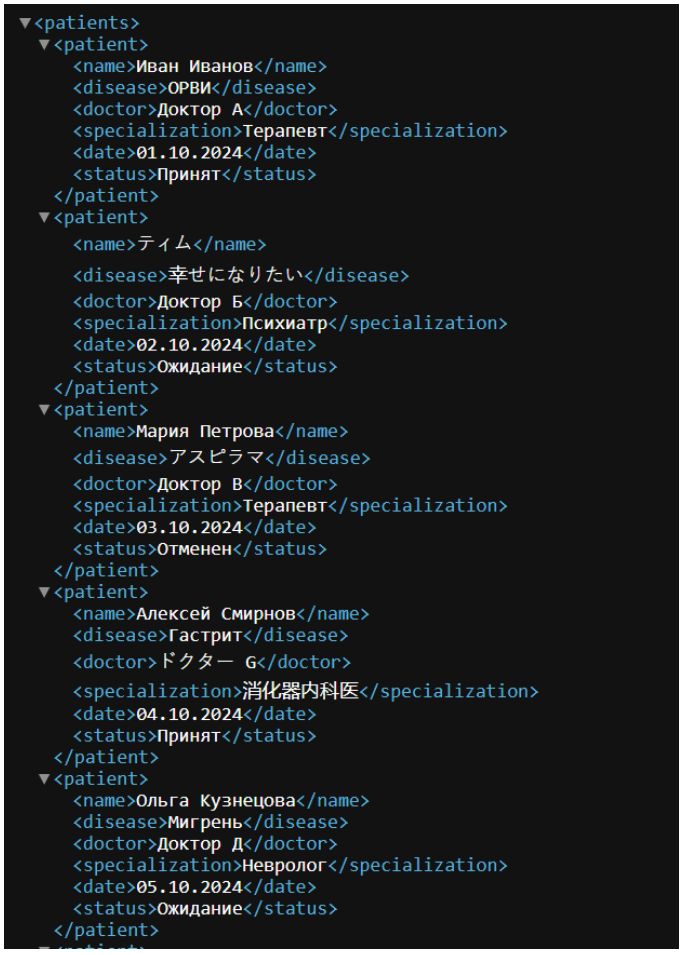
// searchButton

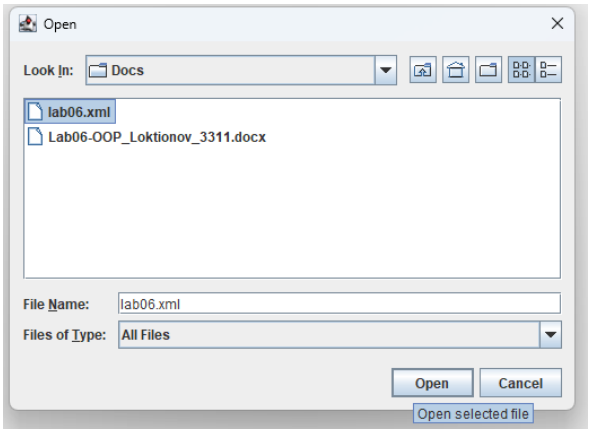


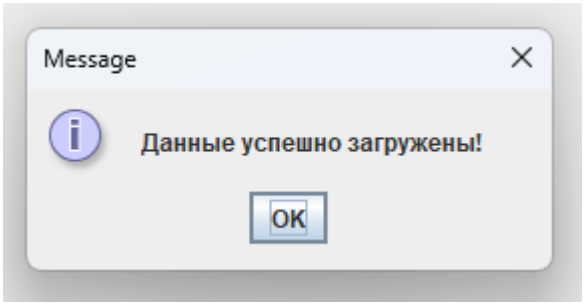
// addButton

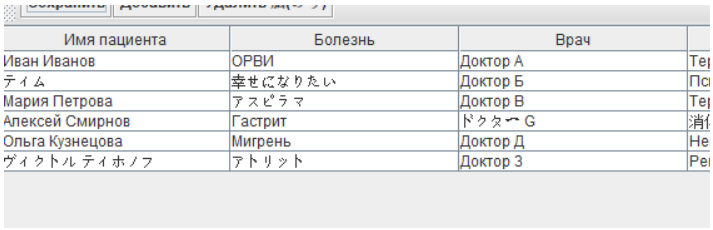


Работоспособность загрузки/выгрузки данных из XML файла:

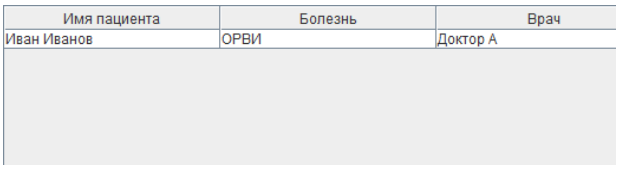




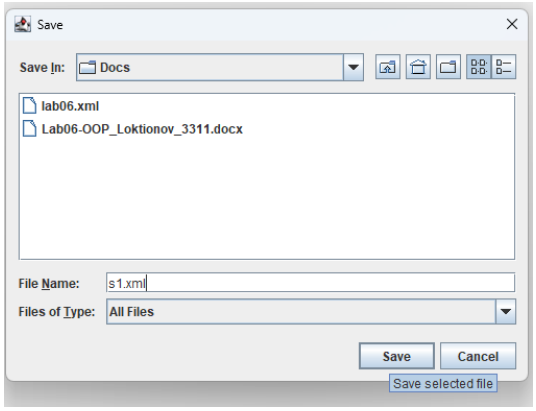


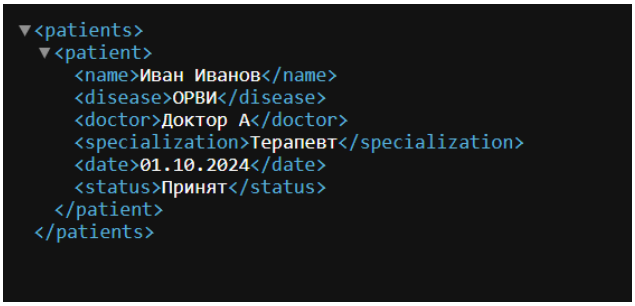


Имя пациента	Болезнь	Врач	
Иван Иванов	ОРВИ	Доктор А	Тер
ティム	幸せになりたい	Доктор Б	Пси
Мария Петрова	アスピラマ	Доктор В	Тер
Алексей Смирнов	Гастрит	ドクター G	消化
Ольга Кузнецова	Мигрень	Доктор Д	Не
ヴィクトル ティホノフ	アトリット	Доктор З	Ре



Имя пациента	Болезнь	Врач
Иван Иванов	ОРВИ	Доктор А





Ссылки:

>> репозиторий:
https://github.com/iconlti/ltprojects/tree/master/oop/java%20projects/hospital-lab06

>> видео отчет:
Google Disk:
https://drive.google.com/drive/folders/1yrkoxpkxttlnzto8e_p5ospojcbnft9a?usp=sharing

Текст программы:

```
// ClinicApp.java
```

```java
import javax.swing.SwingUtilities;

/**
 * Основной класс приложения, содержащий точку входа.
 * @author Tim Loktionov 3311
 * @version 1.00
 */
public class ClinicApp {
    /**
     * Главный метод запуска программы.
     * Вызывает метод создания и отображения интерфейса.
     *
     * @param args аргументы командной строки (не используются).
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new GUI().buildAndShowGUI());
    }
}
```

```
// GUI.java
```

```java
import javax.swing.*;
import java.awt.*;
import java.io.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellRenderer;

/**
 * Основной класс, отвечающий за построение интерфейса приложения "Клиника".
 * Приложение предназначено для управления списком пациентов.
 * Включает добавление, удаление пациентов, сохранение данных и поиск по имени.
 */
public class GUI {
    // Объявление компонентов
    private JFrame frame;
    private JMenuBar menuBar;
    private JMenu fileMenu;
    private JMenuItem openItem, saveItem;
    private JToolBar toolBar;
    private JButton saveButton, addButton, deleteButton;
    private JButton searchButton;
    private JComboBox<String> searchType;
    private JComboBox<String> sortType;
    private JTextField searchField;
    private JTable dataTable;
    private JScrollPane tableScrollPane;
    private DefaultTableModel tableModel;

    /**
     * Метод для построения и отображения графического интерфейса.
     * Создает основное окно приложения, меню, панель инструментов,
     * элементы для поиска и таблицу данных.
     */
    public void buildAndShowGUI() {
        frame = new JFrame("Клиника - Список пациентов");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(1200, 640);

        // Создание меню
        menuBar = new JMenuBar();
        fileMenu = new JMenu("Файл");
        openItem = new JMenuItem("Открыть");
        saveItem = new JMenuItem("Сохранить");
        fileMenu.add(openItem);
        fileMenu.add(saveItem);
        menuBar.add(fileMenu);
        frame.setJMenuBar(menuBar);

        // Панель инструментов
        toolBar = new JToolBar();
        saveButton = new JButton("Сохранить");
        addButton = new JButton("Добавить");
        deleteButton = new JButton("Удалить 脳(のう)");
        toolBar.add(saveButton);
        toolBar.add(addButton);
        toolBar.add(deleteButton);


        // Панель поиска
        JPanel searchPanel = new JPanel();
```

```
        searchType = new JComboBox<>(new String[]{"Ключевому слову", "Имени
пациента", "Имени врача"});
        searchField = new JTextField(25);
        searchButton = new JButton("Поиск");


        // Текст подсказка
        String placeholder = "詳細を入力してください";
        searchField.setText(placeholder);
        searchField.setForeground(Color.RED); // цвет текста

searchField.addFocusListener(Listeners.getSearchFieldFocusListener(searchField,
placeholder));

        searchPanel.add(new JLabel("Поиск по:"));
        searchPanel.add(searchType);
        searchPanel.add(searchField);
        searchPanel.add(searchButton);


        // Контейнер для обеих частей
        JPanel topPanel = new JPanel(new GridLayout(1, 2)); // Одна строка, два
столбца
        topPanel.add(toolBar);
        topPanel.add(searchPanel);
        frame.add(topPanel, BorderLayout.NORTH);


        // Таблица с данными
        String[] columns = {"Имя пациента", "Болезнь", "Врач", "Специализация
врача", "Дата приёма", "Статус"};
        tableModel = new DefaultTableModel(new Object[][]{}, columns);

        dataTable = new JTable(tableModel) {
            @Override
            public Component prepareRenderer(TableCellRenderer renderer, int
row, int column) {
                Component cell = super.prepareRenderer(renderer, row, column);
                // Проверяем, что это колонка "Статус" (индекс 5)
                if (column == 5) {
                    String status = (String) getValueAt(row, column);
                    switch (status) {
                        case "Принят":
                            cell.setBackground(Color.GREEN);
                            break;
                        case "Ожидание":
                            cell.setBackground(Color.YELLOW);
                            break;
                        case "Отменен":
                        case "Отменён":
                            cell.setBackground(Color.RED);
                            break;
                        default:
                            cell.setBackground(Color.WHITE); // Фон для
остальных статусов
                            break;
                    }
                } else {
                    cell.setBackground(Color.WHITE); // Для остальных колонок
устанавливаем белый фон
                }
```

```
                return cell;
            }
        };
        tableScrollPane = new JScrollPane(dataTable);
        frame.add(tableScrollPane, BorderLayout.CENTER);

        // Сортировка
        sortType = new JComboBox<>(new String[]{"По имени", "По дате"});
        frame.add(sortType, BorderLayout.EAST);


        // Слушатели (Action)
        // поиск
        searchButton.addActionListener(Listeners.getSearchListener(dataTable,
searchField, searchType, frame));
        //кнопки
        saveButton.addActionListener(Listeners.getSaveDataListener(frame,
tableModel));

addButton.addActionListener(Listeners.getAddPatientListener(tableModel));

deleteButton.addActionListener(Listeners.getDeletePatientListener(tableModel,
dataTable, frame));
        //сортировка
        sortType.addActionListener(Listeners.getSortTypeActionListener(sortType,
frame));
        // Слушатели для меню
        openItem.addActionListener(Listeners.getLoadDataListener(tableModel,
frame));
        saveItem.addActionListener(Listeners.getSaveToPathDataListener(frame,
tableModel));

        // Визуализация
        frame.setVisible(true);
    }
}
```

// Listeners.java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

/**
 * Класс, содержащий слушатели для различных действий в приложении.
 */
public class Listeners {

    /**
     * Создает слушатель для добавления нового пациента.
     *
     * @param tableModel модель таблицы, в которую будет добавлен пациент
     * @return ActionListener для добавления нового пациента
     */
    public static ActionListener getAddPatientListener(DefaultTableModel
tableModel) {
        return e -> {
            try {
                String name = JOptionPane.showInputDialog("Введите имя
пациента:");
                String disease = JOptionPane.showInputDialog("Введите название
болезни:");
                String doctor = JOptionPane.showInputDialog("Введите имя
врача:");
                String specialization = JOptionPane.showInputDialog("Введите
специализацию врача:");
                String date = JOptionPane.showInputDialog("Введите дату
приёма:");
                String status = JOptionPane.showInputDialog("Введите статус:");

                // Проверка, что поля не пустые и не равны null, так как сама
программа добавляет пустые строки
                if (name != null && !name.trim().isEmpty() &&
                        disease != null && !disease.trim().isEmpty() &&
                        doctor != null && !doctor.trim().isEmpty() &&
                        specialization != null &&
!specialization.trim().isEmpty() &&
                        date != null && !date.trim().isEmpty() &&
                        status != null && !status.trim().isEmpty()) {

                    tableModel.addRow(new Object[]{name, disease, doctor,
specialization, date, status});
                } else {
                    throw new IllegalArgumentException("Все поля должны быть
заполнены!");
                }
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Ошибка добавления пациента:
"
                        + ex.getMessage(), "   エラー",
JOptionPane.ERROR_MESSAGE);
            }
        };
    }
```

```
    /**
     * Создает слушатель для удаления пациента.
     *
     * @param tableModel модель таблицы, из которой будет удален пациент
     * @param dataTable  таблица, отображающая пациентов
     * @param frame      окно, в котором отображаются сообщения
     * @return ActionListener для удаления пациента
     */
    public static ActionListener getDeletePatientListener(DefaultTableModel
tableModel, JTable dataTable, JFrame frame) {
        return e -> {
            try {
                int selectedRow = dataTable.getSelectedRow();
                if (selectedRow != -1) {
                    tableModel.removeRow(selectedRow);
                } else {
                    throw new IllegalArgumentException("Пациент для удаления не
выбран");
                }
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(frame, "Ошибка удаления пациента:
" + ex.getMessage(),
                        "   エラー", JOptionPane.ERROR_MESSAGE);
            }
        };
    }

    /**
     * Создает слушатель для сохранения данных.
     *
     * @param frame окно, в котором отображаются сообщения
     * @return ActionListener для сохранения данных
     */
    public static ActionListener getSaveDataListener(JFrame frame) {
        return e -> JOptionPane.showMessageDialog(frame, "Данные сохранены!");
    }

    /**
     * Создает слушатель для поиска пациента по имени или врачу.
     *
     * @param dataTable   таблица, в которой производится поиск
     * @param searchField поле ввода для текста поиска
     * @param searchType  комбобокс для выбора типа поиска
     * @param frame       окно, в котором отображаются сообщения
     * @return ActionListener для поиска пациента
     */
    public static ActionListener getSearchListener(JTable dataTable, JTextField
searchField,
                                                   JComboBox<String> searchType,
JFrame frame) {
        return e -> {
            try {
                String searchText = searchField.getText().toLowerCase();
                int searchColumn = searchType.getSelectedIndex() == 1 ? 0 : 2;
// 0 - имя пациента, 1 - имя врача

                boolean found = false;
                for (int i = 0; i < dataTable.getRowCount(); i++) {
                    String value = dataTable.getValueAt(i,
searchColumn).toString().toLowerCase();
```

```
                    if (value.contains(searchText)) {
                        dataTable.setRowSelectionInterval(i, i);
                        found = true;
                        break;
                    }
                }
                if (!found) {
                    throw new IllegalArgumentException("Ничего не найдено バカ");
                }
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(frame, "Ошибка поиска: " +
ex.getMessage(),
                        "   エラー", JOptionPane.ERROR_MESSAGE);
            }

        };
    }

    /**
     * Создает слушатель для сортировки пациентов.
     *
     * @param sortType комбобокс для выбора типа сортировки
     * @param frame    окно, в котором отображаются сообщения
     * @return ActionListener для сортировки пациентов
     */
    public static ActionListener getSortTypeActionListener(JComboBox<String>
sortType, JFrame frame) {
        return e -> {
            String selectedSort = (String) sortType.getSelectedItem();
            if ("По имени".equals(selectedSort)) {
                JOptionPane.showMessageDialog(frame, "Сортировка по имени");
            } else if ("По дате".equals(selectedSort)) {
                JOptionPane.showMessageDialog(frame, "Сортировка по дате");
            }
        };
    }

    /**
     * Создает слушатель для управления поведением поля поиска.
     *
     * @param searchField поле ввода для текста поиска
     * @param placeholder текст-заполнитель для поля поиска
     * @return FocusAdapter для управления поведением поля поиска
     */
    public static FocusAdapter getSearchFieldFocusListener(JTextField
searchField, String placeholder) {
        return new FocusAdapter() {
            @Override
            public void focusGained(FocusEvent e) {
                if (searchField.getText().equals(placeholder)) {
                    searchField.setText("");
                    searchField.setForeground(Color.BLACK);
                }
            }

            @Override
            public void focusLost(FocusEvent e) {
                if (searchField.getText().isEmpty()) {
                    searchField.setForeground(Color.RED);
                    searchField.setText(placeholder);
```

```
                    }
                }
            };
        }


    // Work with file //
public static ActionListener getLoadDataListener(DefaultTableModel tableModel, JFrame frame) {
    return e -> {
        JFileChooser fileChooser = new JFileChooser(); // Окно для выбора файла
        int result = fileChooser.showOpenDialog(frame); // Открытие диалогового окна для
выбора файла
        if (result == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile(); // Получаем выбранный файл
            XMLfile.loadFromXML(tableModel, file); // Загружаем данные из файла с помощью
XMLReader
            JOptionPane.showMessageDialog(frame, "Данные успешно загружены!"); // Показываем
сообщение об успехе
        }
    };
}

public static ActionListener getSaveToPathDataListener(JFrame frame, DefaultTableModel
tableModel) {
    return e -> {
        JFileChooser fileChooser = new JFileChooser(); // Окно для выбора пути сохранения
        int result = fileChooser.showSaveDialog(frame); // Открытие диалогового окна для
сохранения файла
        if (result == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile(); // Получаем выбранный файл
            XMLfile.saveToXML(tableModel, file); // Сохраняем данные в файл с помощью
XMLWriter
            JOptionPane.showMessageDialog(frame, "Данные успешно сохранены!"); // Показываем
сообщение об успехе
        }
    };
}
```

// XMLfile.java

```
import org.w3c.dom.*; // объектная модель документа (DOM) --> создание объектов
import javax.xml.parsers.*; // для создания парсеров
// ParserConfigurationException; -- Для обработки ошибок конфигурации парсера ^
import java.io.File; // для работы с файлами
import javax.swing.table.DefaultTableModel;
import javax.xml.transform.*; // для преобразования и записи в XML-документ
import javax.xml.transform.dom.DOMSource; // источник данных DOM для записи
import javax.xml.transform.stream.StreamResult; // класс для записи XML в файл (поток)


public class XMLfile {

    /**
     * Метод для загрузки данных из XML-файла и добавления их в таблицу.
     * @param tableModel модель таблицы, куда будут добавлены данные
     * @param file файл XML, откуда будут загружены данные
     */
    public static void loadFromXML(DefaultTableModel tableModel, File file) {
        try {
            // фабрика для создания парсеров
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

            // создаем сам парсер (builder)
            DocumentBuilder builder = factory.newDocumentBuilder();

            Document doc = builder.parse(file); // загружаем и парсим сам файл
            doc.getDocumentElement().normalize(); // нормализуем "чистим после парсинга"

            NodeList nodeList = doc.getElementsByTagName("patient"); // загружаем все узлы
пациентов (каждый 'patient')
            tableModel.setRowCount(0); // очищаем таблицу перед загрузкой данных

            // проходимся по каждому элементу patients
            for (int i = 0; i < nodeList.getLength(); i++) {
                Node node = nodeList.item(i); // проходимся по каждому узлу
                if (node.getNodeType() == Node.ELEMENT_NODE) {
                    Element element = (Element) node; // преобразуем каждый узел для
возможности работать с аттрибутами

                    // Извлекаем данные из каждого элемента <patient>
                    String name =
element.getElementsByTagName("name").item(0).getTextContent();
                    /* getElementsByTagName("name") -- ищем дочерние элемнты с определенным
именем
                    *  .item(0) -- берем первый элемент из списка
                    *  */
                    String disease =
element.getElementsByTagName("disease").item(0).getTextContent();
                    String doctor =
element.getElementsByTagName("doctor").item(0).getTextContent();
                    String specialization =
element.getElementsByTagName("specialization").item(0).getTextContent();
                    String date =
element.getElementsByTagName("date").item(0).getTextContent();
                    String status =
element.getElementsByTagName("status").item(0).getTextContent();

                    // Добавляем данные в модель таблицы
                    tableModel.addRow(new Object[]{name, disease, doctor, specialization,
date, status});
                }
            }

        } catch (Exception ex) { // доделать, чтобы делал полноценный вывод; ??
            ex.printStackTrace();
        }
    }

    /**
```

```
     * Метод для сохранения данных из таблицы в XML-файл.
     * @param tableModel модель таблицы, из которой будут извлечены данные
     * @param file файл, куда будет записан XML
     */
    public static void saveToXML (DefaultTableModel tableModel, File file) {
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.newDocument();
            Element root = doc.createElement("patients"); // корневой элемент
            doc.appendChild(root);

            // проходим по строкам таблицы и создаем элементы <patient>
            for (int row = 0; row < tableModel.getRowCount(); row++) {
                Element patient = doc.createElement("patient");

                // создаем элементы <name>, <disease>, <doctor> и т.д. для каждого пациента
                Element name = doc.createElement("name");
                name.appendChild(doc.createTextNode(tableModel.getValueAt(row,
0).toString()));
                patient.appendChild(name);

                Element disease = doc.createElement("disease");
                disease.appendChild(doc.createTextNode(tableModel.getValueAt(row,
1).toString()));
                patient.appendChild(disease);

                Element doctor = doc.createElement("doctor");
                doctor.appendChild(doc.createTextNode(tableModel.getValueAt(row,
2).toString()));
                patient.appendChild(doctor);

                Element specialization = doc.createElement("specialization");
                specialization.appendChild(doc.createTextNode(tableModel.getValueAt(row,
3).toString()));
                patient.appendChild(specialization);

                Element date = doc.createElement("date");
                date.appendChild(doc.createTextNode(tableModel.getValueAt(row,
4).toString()));
                patient.appendChild(date);

                Element status = doc.createElement("status");
                status.appendChild(doc.createTextNode(tableModel.getValueAt(row,
5).toString()));
                patient.appendChild(status);

                // добавляем <patient> к корневому элементу <patients>
                root.appendChild(patient);
            }

            TransformerFactory transformerFactory = TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(file);

            // Записываем XML в указанный файл
            transformer.transform(source, result);

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```