

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 № 10
«Протоколирование работы приложения»

Выполнил: Мохно Д. А.

Факультет КТИ

Группа № 3312

Преподаватель: Павловский М.Г.

Подпись преподавателя _____

Санкт-Петербург

2024 г

Цель работы

Знакомство с методами протоколирования работы приложения с использованием библиотеки Log4j.

Перечень используемых типов сообщений, которые выводятся в лог-файл.

- *INFO*: Успешные операции, например, "Файл открыт".
- *DEBUG*: Детальная информация, например, "Список строк: ...".
- *WARN*: Предупреждения, например, "Файл не найден".

Конфигурационный файл log4j.properties.

```
# Устанавливаем уровень логирования для корневого логгера
log4j.rootLogger=INFO, console, file

# Настройка вывода логов в консоль
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{ISO8601} [%t] %-5p %c - %m%n

# Настройка вывода логов в файл
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=application.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ISO8601} [%t] %-5p %c - %m%n
```

Лог-файлы работы приложения в режимах WARN+INFO и DEBUG.

```
2024-12-14 16:18:25,094 [AWT-EventQueue-0] INFO Processor - Открытие файла:
2024-12-14 16:18:36,835 [AWT-EventQueue-0] INFO UserInterface - Удаление1
строка: [5]
2024-12-14 16:18:41,938 [AWT-EventQueue-0] INFO UserInterface - Удаление3
строка: [5, 4, 3]
2024-12-14 16:18:43,270 [AWT-EventQueue-0] INFO UserInterface - Добавлена новая
строка.
2024-12-14 16:18:48,695 [AWT-EventQueue-0] WARN UserInterface - Попытка поиска
без ввода текста.
2024-12-14 16:19:02,690 [AWT-EventQueue-0] INFO UserInterface - Поиск текста:
Мигрень
2024-12-14 16:19:15,562 [AWT-EventQueue-0] INFO UserInterface - Поиск текста:
Волков
2024-12-14 16:19:26,297 [AWT-EventQueue-0] INFO Processor - Экспорт данных в
формат: html
```

```
2024-12-14 16:26:44,990 [AWT-EventQueue-0] INFO Processor - Открытие файла:
2024-12-14 16:26:51,010 [AWT-EventQueue-0] DEBUG Processor - Данные успешно
загружены из файла: /Users/daniilmohno/Library/Mobile
Documents/com~apple~CloudDocs/Student-
staff/ооп/Лабы/com.study_oop.Laba_10/data.xml
2024-12-14 16:26:56,362 [AWT-EventQueue-0] WARN UserInterface - Попытка удалить
строку без выбора.
```

```
2024-12-14 16:27:08,614 [AWT-EventQueue-0] INFO    UserInterface - Удаление2
строк: [1, 0]
2024-12-14 16:27:09,310 [AWT-EventQueue-0] INFO    UserInterface - Добавлена новая
строка.
2024-12-14 16:27:09,310 [AWT-EventQueue-0] DEBUG   UserInterface - Общее
количество строк после добавления: 9
2024-12-14 16:27:17,578 [AWT-EventQueue-0] WARN    UserInterface - Попытка поиска
без ввода текста.
```

Исходные тексты классов, где осуществляется протоколирование работы приложения.

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.*;
import java.util.HashMap;
import java.util.Map;

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 net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.query.JRXPathQueryExecuterFactory;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.PropertyConfigurator;

/**
 * Класс Processor выполняет операции по работе с файлами и таблицами
 * для сохранения, открытия, редактирования и отображения данных в формате
 * CSV.
 */
public class Processor {
    /**
     * Основное окно приложения
     */
    public JFrame window;

    /**
     * Таблица для отображения данных
     */
    public JTable table;

    /**
     * Модель таблицы для управления данными
     */
    public DefaultTableModel tableModel;
```

```

/**
 * Текущий путь к файлу
 */
public String currentFilePath = "";

private static final Logger log = Logger.getLogger(Processor.class);

/**
 * Конструктор, принимающий основное окно приложения.
 *
 * @param frame основное окно приложения JFrame
 */
Processor(JFrame frame) {

PropertyConfigurator.configure("src/main/resources/log4j.properties"); //
Укажите путь к вашему файлу конфигурации
    window = frame;
}

/**
 * Открывает диалоговое окно для выбора CSV-файла и загружает данные в
таблицу.
 */
public void openFile() {
    log.info("Открытие файла: " + currentFilePath);
    FileDialog fd = new FileDialog(window, "Открыть",
java.awt.FileDialog.LOAD);
    fd.setFile(".xml");
    fd.setDirectory(System.getProperty("user.dir"));
    fd.setVisible(true);
    currentFilePath = fd.getDirectory() + fd.getFile();
    if (currentFilePath.endsWith(".xml")) {
        Document data = Parser.parse(currentFilePath);
        tableModel.setRowCount(0); // Очищает таблицу перед загрузкой
новых данных
        tableFill(data);
    } else {
        String[][] text = readFile(currentFilePath);
        tableModel.setRowCount(0); // Очищает таблицу перед загрузкой
новых данных
        tableFill(text);
    }
    log.debug("Данные успешно загружены из файла: " + currentFilePath);
}

/**
 * Сохраняет данные таблицы в текущий CSV/XML-файл.
 */
public void saveFile() {
    if (currentFilePath.isEmpty()) {
        log.warn("Попытка сохранить файл без указания пути.\nСохраняем в
новый файл");
        saveFileAs();
    }
    else if (currentFilePath.endsWith(".xml")) {
        log.info("Сохранение данных формата xml в файл: " +
currentFilePath);
        writeFile(Parser.to_xml(tableModel));
    }
    else {
        log.info("Сохранение данных формата csv в файл: " +
currentFilePath);
    }
}

```

```

        writeFile(currentFilePath);
    }
}

/**
 * Открывает диалоговое окно для выбора нового места и имени для
 * сохранения CSV-файла.
 */
public void saveFileAs() {
    FileDialog fd = new FileDialog(window, "Сохранить как",
    java.awt.FileDialog.SAVE);
    fd.setFile("Untitled.xml");
    fd.setDirectory(System.getProperty("user.home"));
    fd.setVisible(true);
    String path = fd.getDirectory() + fd.getFile();
    if (!path.equals("nullnull")) {
        currentFilePath = path;
        if (path.endsWith(".xml")) writeFile(Parser.to_xml(tableModel));
        else writeFile(currentFilePath);
    }
}

/**
 * Выводит файл на печать.
 */
public void printFile() {
    try {
        JasperPrint print = getReport(Parser.to_xml(tableModel));
        if (print != null)
            JasperPrintManager.printReport(print, true);
        else
            JOptionPane.showMessageDialog(window, "Ошибка при
формировании отчёта", "ERROR", JOptionPane.INFORMATION_MESSAGE);
    } catch (JRException e) {
        e.printStackTrace();
    }
}

/**
 * Экспортирует файлы.
 *
 * @param format Формат файла "html" или "pdf"
 */
public void exportFile(String format) {
    log.info("Экспорт данных в формат: " + format);
    // Этап 1: создание документа XML в отдельном потоке
    Thread loadDataThread = new Thread(() -> {
        try {
            Document doc = Parser.to_xml(tableModel);
            // Этап 2: подготовка отчёта в другом потоке
            Thread generateReportThread = new Thread(() -> {
                JasperPrint print = getReport(doc);
                if (print != null) {
                    // Этап 3: экспорт отчёта в файл в третьем потоке
                    Thread exportFileThread = new Thread(() -> {
                        try {
                            FileDialog fd = new FileDialog(window,
                            "", FileDialog.SAVE);
                            fd.setDirectory(System.getProperty("user.dir"));
                            fd.setFile("Untitled." + format);
                            fd.setVisible(true);

```

```

        String output_file_path =
        fd.getDirectory() + fd.getFile();
        if (output_file_path.endsWith("pdf")) {
            File output_file = new
            File(output_file_path);

            JasperExportManager.exportReportToPdfFile(print,
            output_file.getAbsolutePath());
        } else if
        (output_file_path.endsWith("html")) {
            File output_file = new
            File(output_file_path);

            JasperExportManager.exportReportToHtmlFile(print,
            output_file.getAbsolutePath());
        }
        log.debug("Отчёт успешно создан для
        экспорта в формат " + format);
    } catch (JRException e) {
        log.error("Ошибка при создании отчёта для
        экспорта", e);
    }
    });
    exportFileThread.start(); // Запуск потока
    экспорта
    } else JOptionPane.showMessageDialog(window, "Ошибка
    при формировании отчёта", "ERROR", JOptionPane.INFORMATION_MESSAGE);
    });
    generateReportThread.start(); // Запуск потока генерации
    отчёта
    } catch (Exception e) {
        e.printStackTrace();
    }
    });
    loadDataThread.start(); // Запуск потока загрузки данных
}

/**
 * Конвертирует таблицу в объект для печати или экспорта
 *
 * @param doc объект Document содержащий xml таблицу
 * @return объект JasperPrint для экспорта или печати или null в случае
    ошибки
 */
private JasperPrint getReport(Document doc) {
    try {
        String template = "MyReports/data.jrxml";
        JasperReport jasperReport =
        JasperCompileManager.compileReport(template);
        Map params = new HashMap();

        params.put(JRXPathQueryExecuterFactory.PARAMETER_XML_DATA_DOCUMENT, doc);
        return JasperFillManager.fillReport(jasperReport, params);
    } catch (JRException e) {
        e.printStackTrace();
    }
    return null;
}

/**
 * Читает содержимое CSV-файла и возвращает его как двумерный массив
    строк.

```

```

*
* @param pathToFile путь к файлу для чтения
* @return двумерный массив строк, представляющий данные таблицы,
* или null в случае ошибки чтения файла
*/
public String[][] readFile(String pathToFile) {
    try (BufferedReader br = new BufferedReader(new
FileReader(pathToFile))) {
        String[] text = br.lines().toArray(String[]::new);

        String[][] data = new String[text.length][];
        for (int i = 0; i < text.length; i++) {
            String[] line = text[i].split(",", 5);
            data[i] = new String[line.length];
            System.arraycopy(line, 0, data[i], 0, line.length);
        }
        return data;
    } catch (IOException e) {
        JOptionPane.showMessageDialog(window, "Проблема с открытием файла
" + pathToFile + "\n" +
        "Проверьте, существует ли файл", "ERROR",
JOptionPane.INFORMATION_MESSAGE);
        return null;
    }
}

/**
* Записывает содержимое таблицы в CSV-файл.
*
* @param pathToFile путь к файлу для записи
*/
public void writeFile(String pathToFile) {
    try (BufferedWriter br = new BufferedWriter(new
FileWriter(pathToFile))) {
        for (int i = 0; i < table.getRowCount(); i++) {
            for (int j = 0; j < table.getColumnCount(); j++) {
                br.write(tableModel.getValueAt(i, j).toString());
                br.write(",");
            }
            br.newLine();
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(window, "Проблема с сохранением в
файл " + pathToFile,
        "ERROR", JOptionPane.INFORMATION_MESSAGE);
    }
}

/**
* Записывает содержимое таблицы в XML-файл.
*
* @param data объект xml документа
*/
private void writeFile(Document data) {
    try {
        // Создание преобразователя документа
        Transformer trans =
TransformerFactory.newInstance().newTransformer();
        // Создание файла с именем books.xml для записи документа
        java.io.FileWriter fw = new FileWriter(currentFilePath);
        // Запись документа в файл
        trans.transform(new DOMSource(data), new StreamResult(fw));
    }
}

```

```

    } catch (TransformerException | IOException e) {
        JOptionPane.showMessageDialog(window, "Проблема с сохранением в
        файл " + currentFilePath,
            "ERROR", JOptionPane.INFORMATION_MESSAGE);
    }
}

/**
 * Заполняет таблицу данными из двумерного массива строк.
 *
 * @param text двумерный массив строк, представляющий данные для таблицы
 */
private void tableFill(String[][] text) {
    if (text != null) {
        for (String[] s : text) {
            tableModel.addRow(s);
        }
    }
}

/**
 * Заполняет таблицу данными из двумерного массива строк.
 *
 * @param data объект xml документа
 */
private void tableFill(Document data) {
    if (data != null) {
        NodeList list = data.getElementsByTagName("record");
        // Цикл просмотра списка элементов и запись данных в таблицу
        for (int i = 0; i < list.getLength(); i++) {
            // Выбор очередного элемента списка
            Node elem = list.item(i);
            // Получение списка атрибутов элемента
            NamedNodeMap attrs = elem.getAttributes();
            // Чтение атрибутов элемента
            String client = attrs.getNamedItem("client").getNodeValue();
            String doctor = attrs.getNamedItem("doctor").getNodeValue();
            String date = attrs.getNamedItem("date").getNodeValue();
            String time = attrs.getNamedItem("time").getNodeValue();
            String symptoms =
attrs.getNamedItem("symptoms").getNodeValue();
            // Запись данных в таблицу
            tableModel.addRow(new String[]{client, doctor, date, time,
symptoms});
        }
    }
}

/**
 * Создает таблицу с заданными именами столбцов.
 *
 * @param colNames массив строк с именами столбцов
 */
public void createTable(String[] colNames) {
    tableModel = new DefaultTableModel(null, colNames);
    table = new JTable(tableModel);
}

/**
 * Класс Parser предоставляет методы для работы с XML-файлами.
 * Содержит статические методы для чтения данных из XML-файла в объект
 * Document`

```



```

    * и для преобразования данных таблицы в XML-формат.
    */
    static class Parser {
        /**
         * Парсит XML-файл и возвращает объект `Document`.
         *
         * @param pathToFile путь к XML-файлу для парсинга
         * @return объект `Document`, представляющий содержимое XML-файла,
         * или null в случае ошибки чтения или парсинга
         */
        public static Document parse(String pathToFile) {
            try {
                // Создание парсера документа
                DocumentBuilder dBuilder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
                // Чтение документа из файла
                Document doc = dBuilder.parse(new File(pathToFile));
                // Нормализация документа
                doc.getDocumentElement().normalize();
                return doc;
            } catch (ParserConfigurationException | IOException |
SAXException e) {
                JOptionPane.showMessageDialog(null, "Проблема с чтением файла
" + pathToFile,
                    "ERROR", JOptionPane.INFORMATION_MESSAGE);
            }
            // Обработка ошибки парсера при чтении данных из XML-файла
            return null;
        }

        /**
         * Преобразует данные из таблицы в XML-формат и возвращает объект
         `Document`.
         *
         * @param tableModel модель таблицы `DefaultTableModel`, содержащая
         данные для преобразования
         * @return объект `Document`, представляющий данные в XML-формате
         * @throws RuntimeException если возникает ошибка конфигурации
         парсера
         */
        public static Document to_xml(DefaultTableModel tableModel) {
            try {
                // Создание парсера документа
                DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
                // Создание пустого документа
                Document doc = builder.newDocument();

                // Создание корневого элемента booklist и добавление его в
документ

                Node data = doc.createElement("base");
                doc.appendChild(data);

                // Создание дочерних элементов book и присвоение значений
атрибутам

                for (int i = 0; i < tableModel.getRowCount(); i++) {
                    Element book = doc.createElement("record");
                    data.appendChild(book);
                    book.setAttribute("client", (String)
tableModel.getValueAt(i, 0));
                    book.setAttribute("doctor", (String)
tableModel.getValueAt(i, 1));
                }
            }
        }
    }

```

```

        book.setAttribute("date", (String)
tableModel.getValueAt(i, 2));
        book.setAttribute("time", (String)
tableModel.getValueAt(i, 3));
        book.setAttribute("symptoms", (String)
tableModel.getValueAt(i, 4));
    }
    return doc;
} catch (ParserConfigurationException e) {
    throw new RuntimeException(e);
}
}
}
}
}

```

```

/**
 * Лабораторная работа №8
 *
 * @author Даниил Мохно 3312
 * @version 1.0
 */

import javax.swing.*;
import java.awt.*;
import java.awt.Taskbar;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusAdapter;
import java.awt.event.FocusEvent;

import org.apache.log4j.Logger;

/**
 * Основной пользовательский интерфейс приложения, включает элементы окна,
 * панелей, кнопок, меню и таблицы для отображения и взаимодействия с
 * данными.
 */
public class UserInterface {
    /**
     * Главное окно приложения.
     */
    public JFrame window;

    /**
     * Иконка приложения для отображения в заголовке окна.
     */
    private Image icon;

    /**
     * Главная панель для размещения всех компонентов.
     */
    private JPanel panel;

    /**
     * Верхняя панель интерфейса, используется для меню и поиска.
     */
    private JPanel header;

    /**

```

```
    * Панель для размещения кнопки сохранения.
    */
    private JPanel saveArea;

    /**
     * Панель для размещения поля и кнопки поиска.
     */
    private JPanel searchArea;

    /**
     * Кнопка сохранения файла.
     */
    public JButton saveButton;

    /**
     * Строка меню, содержащая основные действия приложения.
     */
    public JMenuBar menuBar;

    /**
     * Поле поиска для ввода текста поиска.
     */
    public JTextField searchField;

    /**
     * Кнопка для выполнения поиска по введённому тексту.
     */
    public JButton searchButton;

    /**
     * Текст-заполнитель для поля поиска.
     */
    public String placeholder;

    /**
     * Нижняя панель интерфейса, используется для кнопок добавления и
    справки.
     */
    private JPanel footer;

    /**
     * Кнопка для отображения справочной информации.
     */
    public JButton referenceButton;

    /**
     * Кнопка для добавления новой строки в таблицу.
     */
    public JButton add;

    /**
     * Кнопка для удаления выбранной строки из таблицы.
     */
    public JButton del;

    /**
     * Панель для размещения кнопок добавления и удаления.
     */
    private JPanel editPanel;

    /**
     * Модель таблицы, управляющая данными, отображаемыми в таблице.

```

```

    */
    public DefaultTableModel tableModel;

    /**
     * Таблица для отображения данных.
     */
    public JTable table;

    /**
     * Панель прокрутки для таблицы, позволяет прокручивать данные при
     * большом объеме.
     */
    public JScrollPane scrollPane;

    /**
     * Обработчик для работы с файлами и таблицей.
     */
    public Processor processor;

    private static final Logger log = Logger.getLogger(UserInterface.class);

    /**
     * Метод для отображения окна приложения
     */
    public void show() {
        if (System.getProperty("os.name").toLowerCase().contains("mac"))
            System.setProperty("apple.awt.application.name", "Поликлиника");

        // Создание основного окна
        window = new JFrame("Поликлиника");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setSize(800, 400);
        window.setLocationRelativeTo(null);

        // Добавление обработчика
        processor = new Processor(window);

        // Добавление иконки
        icon =
window.getToolkit().getImage(getClass().getResource("/img/icon.png"));
        if (System.getProperty("os.name").toLowerCase().contains("mac")) {
            Taskbar.getTaskbar().setIconImage(icon);
            System.setProperty("apple.laf.useScreenMenuBar", "true");
        } else {
            window.setIconImage(icon);
        }

        // Устанавливаем панель
        panel = new JPanel();
        panel.setBackground(new Color(216, 240, 211));
        panel.setLayout(new BorderLayout());

        // Создание меню
        menuBar = new JMenuBar();
        String[] menus = {"File", "Edit", "View", "Help"};
        String[][] menuItems = {"Open file", "Save", "Save as", "Print"},
{"Add", "Delete"};
        Runnable[][] events = {{
            processor::openFile,
            processor::saveFile,
            processor::saveFileAs,
            processor::printFile,

```

```

    }, {this::addRow, this::delRow}};
    for (int i = 0; i < menus.length; i++) {
        JMenu menu = new JMenu(menus[i]);
        for (int j = 0; menuItems.length > i && j < menuItems[i].length;
j++) {
            JMenuItem item = new JMenuItem(menuItems[i][j]);
            final int menuIndex = i;
            final int itemIndex = j;
            item.addActionListener(e ->
events[menuIndex][itemIndex].run());
            menu.add(item);
        }
        menuBar.add(menu);
    }
    // Добавление меню экспорта
    JMenu export = new JMenu("Export");
    JMenuItem html = new JMenuItem("HTML");
    html.addActionListener(e -> processor.exportFile("html"));
    JMenuItem pdf = new JMenuItem("PDF");
    pdf.addActionListener(e -> processor.exportFile("pdf"));
    export.add(html);
    export.add(pdf);
    menuBar.getMenu(0).add(export);

    // Создаём шапку
    header = new JPanel();
    header.setBackground(new Color(211, 240, 228));
    header.setLayout(new BorderLayout());

    // Кнопка сохранения
    saveArea = new JPanel();
    saveArea.setBackground(new Color(211, 240, 228));
    saveButton = new JButton("Сохранить");
    saveArea.add(saveButton);
    header.add(saveArea, BorderLayout.WEST);

    // Добавление меню
    window.setJMenuBar(menuBar);

    // Поле поиска
    searchField = getSearchField();

    // Кнопка поиска
    searchButton = new JButton("Искать");
    searchButton.setPreferredSize(new Dimension(100, 30));

    // Добавляем поиск в шапку
    searchArea = new JPanel();
    searchArea.add(searchButton, BorderLayout.EAST);
    searchArea.setBackground(new Color(211, 240, 228));
    searchArea.add(searchField, BorderLayout.EAST);
    header.add(searchArea, BorderLayout.EAST);

    // Добавляем шапку на панель
    panel.add(header, BorderLayout.NORTH);

    // создаём футер
    footer = new JPanel();
    footer.setBackground(new Color(211, 240, 228));
    footer.setLayout(new BorderLayout());

```

```

// Кнопка показа справки
referenceButton = new JButton("Справка");
referenceButton.setPreferredSize(new Dimension(100, 30));
footer.add(referenceButton, BorderLayout.WEST);

// Кнопка добавления строки в таблицу
add = new JButton("+");
del = new JButton("-");
editPanel = new JPanel();
editPanel.setBackground(new Color(211, 240, 228));
editPanel.setLayout(new FlowLayout(FlowLayout.LEFT, 0, 0));
editPanel.add(del);
editPanel.add(add);

footer.add(editPanel, BorderLayout.EAST);
panel.add(footer, BorderLayout.SOUTH);

// Создаём табличку
String[] columnNames = {"ФИО ПАЦИЕНТА", "ФИО ВРАЧА", "ДАТА ЗАПИСИ",
"ВРЕМЯ ЗАПИСИ", "ЖАЛОБЫ"};
processor.createTable(columnNames);
table = processor.table;
tableModel = processor.tableModel;

// Оформляем таблицу
table.getTableHeader().setBackground(new Color(100, 180, 100)); //
зелёный заголовок
table.getTableHeader().setForeground(Color.WHITE);
table.setRowHeight(30);
table.getTableHeader().setReorderingAllowed(false); // Запрет менять
местами
table.setGridColor(new Color(19, 54, 11));
table.setSelectionBackground(new Color(216, 240, 211));
table.setSelectionForeground(new Color(4, 17, 3));

// Добавляем скролл
scrollPane = new JScrollPane(table);
panel.add(scrollPane, BorderLayout.CENTER);

// Добавление панели в основное окно
window.add(panel);

add_filters();

// Отображение окна
window.setVisible(true);
}

/**
 * Создаёт поле поиска с автоматически убирающимся текстом внутри
 *
 * @return поле поиска
 */
private JTextField getSearchField() {
    searchField = new JTextField(15);
    searchField.setPreferredSize(new Dimension(100, 30));

    // Устанавливаем начальный текст как плейсхолдер
    placeholder = "🔍 Поиск";
    searchField.setText(placeholder);
    searchField.setForeground(Color.GRAY);
}

```

```

// Добавляем FocusListener для обработки placeholder
searchField.addFocusListener(new FocusAdapter() {
    /**
     * Убирает плейсхолдер при фокусировании на поле
     * @param e событие для обработки
     */
    @Override
    public void focusGained(FocusEvent e) {
        if (searchField.getText().equals(placeholder)) {
            searchField.setText("");
            searchField.setForeground(Color.BLACK); // Обычный цвет
текста
        }
    }

    /**
     * Добавляет плейсхолдер при расфокусировании из поля
     * @param e событие для обработки
     */
    @Override
    public void focusLost(FocusEvent e) {
        if (searchField.getText().isEmpty()) {
            searchField.setText(placeholder);
            searchField.setForeground(Color.GRAY); // Цвет
placeholder
        }
    }
});
return searchField;
}

/**
 * Добавляет обработчики событий к элементам управления.
 */
private void add_filters() {
    JComponent[] fields = {add, del, searchButton, saveButton,
referenceButton};
    Runnable[] func = {this::addRow, this::delRow, this::search,
processor::saveFile, this::show_ref};

    for (int i = 0; i < fields.length; i++) {
        final int index = i;
        // Добавляем слушатель
        ((AbstractButton) fields[i]).addActionListener(new
ActionListener() {
            /** Добавляем действие */
            @Override
            public void actionPerformed(ActionEvent e) {
                func[index].run();
            }
        });
    }
}

/**
 * Добавление строки в таблицу
 */
private void addRow() {
    tableModel.addRow(new Object[]{"", "", "", "", ""});
    log.info("Добавлена новая строка.");
    log.debug("Общее количество строк после добавления: " +

```

```

tableModel.getRowCount());
    }

    /**
     * Удаление строки из таблицы
     */
    private void delRow() {
        if (table.getSelectedRow() != -1) {
            log.info("Удаление" + getSelectedRows().length + " строк: " +
java.util.Arrays.toString(getSelectedRows()));
            for (int i : getSelectedRows())
                tableModel.removeRow(i);
        } else {
            log.warn("Попытка удалить строку без выбора.");
        }
    }

    /**
     * Получает выделенные строки таблицы и возвращает их в перевёрнутом
    порядке.
     *
     * @return перевёрнутый массив выделенных строк таблицы
     */
    private int[] getSelectedRows() {
        int[] s_rows = table.getSelectedRows();
        for (int i = 0; i < s_rows.length / 2; i++) {
            int temp = s_rows[i];
            s_rows[i] = s_rows[s_rows.length - 1 - i];
            s_rows[s_rows.length - 1 - i] = temp;
        }
        return s_rows;
    }

    /**
     * Выполняет поиск по тексту, введенному в поле поиска,
     * и подсвечивает строки, которые содержат этот текст.
     */
    private void search() {
        String searchText = searchField.getText().trim();

        // Проверка на пустой текст (или placeholder)
        try {
            NotSearched.checkString(searchText, placeholder);
            log.info("Поиск текста: " + searchField.getText());
        } catch (NotSearched e) {
            log.warn("Попытка поиска без ввода текста.");
            return; // Возвращаемся, если текст пустой или совпадает с
плейсхолдером
        } // Если текст пустой или placeholder, выводим предупреждение

        boolean found = false; // Инициализация переменной после проверки
пустого ввода

        // Перебираем все строки таблицы
        for (int i = 0; i < table.getRowCount(); i++) {
            boolean rowContainsSearchText = false;

            // Проверяем каждую ячейку строки на совпадение с текстом поиска
            for (int j = 0; j < table.getColumnCount(); j++) {
                Object cellValue = table.getValueAt(i, j);
                if (cellValue != null &&

```



```

cellValue.toString().toLowerCase().contains(searchText.toLowerCase())) {
    rowContainsSearchText = true;
    break;
}
}

// Если строка содержит текст, подсвечиваем её
if (rowContainsSearchText) {
    found = true;
    table.setRowSelectionInterval(i, i); // Подсвечиваем строку
    table.scrollRectToVisible(table.getCellRect(i, 0, true)); //
Прокручиваем к строке
    break; // Выход из цикла после нахождения первой строки
}
}

// Если текст не найден, показываем сообщение
if (!found) {
    JOptionPane.showMessageDialog(window, "Текст не найден.");
}
}

/**
 * Отображает справочную информацию.
 */
private void show_ref() {
    if (table.getSelectedRow() != -1) {
        int selectedRow = table.getSelectedRow();
        String fio = (String) table.getValueAt(selectedRow, 0);
        String responsible = (String) table.getValueAt(selectedRow, 1);
        String date = (String) table.getValueAt(selectedRow, 2);
        String symptom = (String) table.getValueAt(selectedRow, 4);

        String reference = "<html>"
            + "<head>"
            + "<style>"
            + "body { font-family: Arial, sans-serif; padding: 20px;"
            + "h1 { text-align: center; }"
            + "p { font-size: 14px; line-height: 1.5; }"
            + "strong { font-weight: bold; }"
            + "table { width: 100%; margin-top: 20px; border-
collapse: collapse; }"
            + "th, td { border: 1px solid #000; padding: 8px; text-
align: left; }"
            + "th { background-color: #f2f2f2; }"
            + "</style>"
            + "</head>"
            + "<body>"
            + "<h1>Медицинская справка</h1>"
            + "<p><strong>ФИО пациента:</strong> " + fio + "</p>"
            + "<p><strong>Дата обращения:</strong> " + date + "</p>"
            + "<p><strong>Жалобы:</strong> " + symptom + "</p>"
            + "<p><strong>Назначения врача:</strong> </p>"
            + "<table>"
            + "<tr><th>Дата</th><th>Назначение</th></tr>"
            + "<tr><td>" + date + "</td><td></td></tr>" // Здесь вы
можете добавить свои назначения
            + "</table>"
            + "<p>Врач " + responsible + "</p>"
            + "<p>Подпись: _____</p>"

```

```

        + "<p>Дата: " + date + "</p>"
        + "</body>";
        JOptionPane.showMessageDialog(null, reference, "Справка",
JOptionPane.INFORMATION_MESSAGE);
    }

    /**
     * Исключение, которое выбрасывается, когда операция поиска выполняется
    без действительного ввода.
     */
    private static class NotSearched extends Exception {
        /**
         * Конструктор для создания исключения NotSearched с сообщением по
        умолчанию.
         */
        public NotSearched() {
            super("Вы не ввели текст для поиска😞");
        }

        /**
         * Проверяет, является ли предоставленная строка пустой или совпадает
        с плейсхолдером.
         *
         * @param str строка для проверки
         * @param placeholder плейсхолдер, указывающий на отсутствие
        действительного ввода
         * @throws NotSearched если строка пустая или равна плейсхолдеру
         */
        private static void checkString(String str, String placeholder)
        throws NotSearched {
            if ((str.isEmpty() || str.equals(placeholder))) {
                throw new NotSearched();
            }
        }

    }

    public static void main(String[] args) {
        UserInterface userInterface = new UserInterface();
        userInterface.show();
    }
}

```

Текст документации, сгенерированный Javadoc.

PACKAGECLASSUSE TREEINDEX SEARCHHELP

Unnamed Package > **UserInterface**

Search

Class UserInterface

java.lang.Object[Ⓢ]
UserInterface

public class **UserInterface**
extends Object[Ⓢ]

Основной пользовательский интерфейс приложения, включает элементы окна, панелей, кнопок, меню и таблицы для отображения и взаимодействия с данными.

Nested Class Summary

Nested Classes

| Modifier and Type | Class | Description |
|----------------------|----------------------------------|-------------------------------------------------------------------------------------------------|
| private static class | UserInterface.NotSearched | Исключение, которое выбрасывается, когда операция поиска выполняется без действительного ввода. |

Field Summary

Fields

| Modifier and Type | Field | Description |
|----------------------------------------------|------------------------|---------------------------------------------------------------------------------|
| JButton [Ⓢ] | add | Кнопка для добавления новой строки в таблицу. |
| JButton [Ⓢ] | del | Кнопка для удаления выбранной строки из таблицы. |
| private JPanel [Ⓢ] | editPanel | Панель для размещения кнопок добавления и удаления. |
| private JPanel [Ⓢ] | footer | Нижняя панель интерфейса, используется для кнопок добавления и справки. |
| private JPanel [Ⓢ] | header | Верхняя панель интерфейса, используется для меню и поиска. |
| private Image [Ⓢ] | icon | Иконка приложения для отображения в заголовке окна. |
| private static final org.apache.log4j.Logger | log | |
| JMenuBar [Ⓢ] | menuBar | Строка меню, содержащая основные действия приложения. |
| private JPanel [Ⓢ] | panel | Главная панель для размещения всех компонентов. |
| String [Ⓢ] | placeholder | Текст-заполнитель для поля поиска. |
| Processor | processor | Обработчик для работы с файлами и таблицей. |
| JButton [Ⓢ] | referenceButton | Кнопка для отображения справочной информации. |
| private JPanel [Ⓢ] | saveArea | Панель для размещения кнопки сохранения. |
| JButton [Ⓢ] | saveButton | Кнопка сохранения файла. |
| JScrollPane [Ⓢ] | scrollPane | Панель прокрутки для таблицы, позволяет прокручивать данные при большом объёме. |
| private JPanel [Ⓢ] | searchArea | Панель для размещения поля и кнопки поиска. |
| JButton [Ⓢ] | searchButton | Кнопка для выполнения поиска по введённому тексту. |
| JTextField [Ⓢ] | searchField | Поле поиска для ввода текста поиска. |
| JTable [Ⓢ] | table | Таблица для отображения данных. |
| DefaultTableModel [Ⓢ] | tableModel | Модель таблицы, управляющая данными, отображаемыми в таблице. |
| JFrame [Ⓢ] | window | Главное окно приложения. |

Constructor Summary [Ⓢ]

Constructors

| Constructor | Description |
|------------------------|-------------|
| UserInterface() | |

Method Summary

All MethodsStatic MethodsInstance MethodsConcrete Methods

| Modifier and Type | Method | Description |
|---------------------------------|--------------------------|----------------------------------------------------------------------------|
| private void | add_filters() | Добавляет обработчики событий к элементам управления. |
| private void | addRow() | Добавление строки в таблицу |
| private void | delRow() | Удаление строки из таблицы |
| private JTextField [Ⓢ] | getSearchField() | Создаёт поле поиска с автоматически убирающимся текстом внутри |
| private int[] | getSelectedRows() | Получает выделенные строки таблицы и возвращает их в перевернутом порядке. |

| | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|----------------------------------------------------------------------------------------------------------|
| private int[] | getSelectedRows() | Получает выделенные строки таблицы и возвращает их в перевернутом порядке. |
| static void | main(String[] args) | |
| private void | search() | Выполняет поиск по тексту, введенному в поле поиска, и подсвечивает строки, которые содержат этот текст. |
| void | show() | Метод для отображения окна приложения |
| private void | show_ref() | Отображает справочную информацию. |
| Methods inherited from class java.lang.Object | | |
| clone , equals , finalize , getClass , hashCode , notify , notifyAll , toString , wait , wait , wait | | |

Field Details

window

public JFrame window

Главное окно приложения.

icon

private Image icon

Иконка приложения для отображения в заголовке окна.

panel

private JPanel panel

Главная панель для размещения всех компонентов.

header

private JPanel header

Верхняя панель интерфейса, используется для меню и поиска.

saveArea

private JPanel saveArea

Панель для размещения кнопки сохранения.

searchArea

private JPanel searchArea

Панель для размещения поля и кнопки поиска.

saveButton

public JButton saveButton

Кнопка сохранения файла.

menuBar

public JMenuBar menuBar

Строка меню, содержащая основные действия приложения.

searchField

public JTextField searchField

Поле поиска для ввода текста поиска.

searchButton

public JButton searchButton

Кнопка для выполнения поиска по введённому тексту.

placeholder

public String placeholder

Текст-заполнитель для поля поиска.

footer

private JPanel footer

Нижняя панель интерфейса, используется для кнопок добавления и справки.

referenceButton

public JButton referenceButton

Кнопка для отображения справочной информации.

add

public JButton add

Кнопка для добавления новой строки в таблицу.

del

public JButton del

Кнопка для удаления выбранной строки из таблицы.

editPanel

```
private JPanel editPanel
```

Панель для размещения кнопок добавления и удаления.

tableModel

```
public DefaultTableModel tableModel
```

Модель таблицы, управляющая данными, отображаемыми в таблице.

table

```
public JTable table
```

Таблица для отображения данных.

scrollPane

```
public JScrollPane scrollPane
```

Панель прокрутки для таблицы, позволяет прокручивать данные при большом объёме.

processor

```
public Processor processor
```

Обработчик для работы с файлами и таблицей.

log

```
private static final org.apache.log4j.Logger log
```

Constructor Details

UserInterface

```
public UserInterface()
```

Method Details

show

```
public void show()
```

Метод для отображения окна приложения

getSearchField

```
private JTextField getSearchField()
```

Создаёт поле поиска с автоматически убирающимся текстом внутри

Returns:

поле поиска

add_filters

```
private void add_filters()
```

Добавляет обработчики событий к элементам управления.

addRow

```
private void addRow()
```

Добавление строки в таблицу

delRow

```
private void delRow()
```

Удаление строки из таблицы

getSelectedRows

```
private int[] getSelectedRows()
```

Получает выделенные строки таблицы и возвращает их в перевёрнутом порядке.

Returns:

перевёрнутый массив выделенных строк таблицы

search

```
private void search()
```

Выполняет поиск по тексту, введенному в поле поиска, и подсвечивает строки, которые содержат этот текст.

show_ref

```
private void show_ref()
```

Отображает справочную информацию.

main

```
public static void main(String[] args)
```

Class Processor

java.lang.Object

```
public class Processor
extends Object
```

Класс Processor выполняет операции по работе с файлами и таблицами для сохранения, открытия, редактирования и отображения данных в формате CSV.

Nested Class Summary

Nested Classes

| Modifier and Type | Class | Description |
|--------------------------------|-------------------------------|-------------------------------------------------------------|
| (package private) static class | <code>Processor.Parser</code> | Класс Parser предоставляет методы для работы с XML-файлами. |

Field Summary

Fields

| Modifier and Type | Field | Description |
|----------------------------------------------|-----------------|---------------------------------------|
| String ¹² | currentFilePath | Текущий путь к файлу |
| private static final org.apache.log4j.Logger | log | |
| JTable ¹³ | table | Таблица для отображения данных |
| DefaultTableModel ¹⁴ | tableModel | Модель таблицы для управления данными |
| JFrame ¹⁵ | window | Основное окно приложения |

Constructor Summary

Constructors

| Constructor | Description |
|--------------------------------------------------|----------------------------------------------------|
| <code>Processor(JFrame^a frame)</code> | Конструктор, принимающий основное окно приложения. |

Method Summary

All Methods

Instance Methods

Concrete Methods

| Modifier and Type | Method | Description |
|-------------------------------------------------|---------------------------------------------|-------------------------------------------------------------------------------------|
| void | <code>createTable(String[] colNames)</code> | Создает таблицу с заданными именами столбцов. |
| void | <code>exportFile(String format)</code> | Экспортирует файлы. |
| private net.sf.jasperreports.engine.JasperPrint | <code>getReport(Document doc)</code> | Конвертирует таблицу в объект для печати или экспорта |
| void | <code>openFile()</code> | Открывает диалоговое окно для выбора CSV-файла и загружает данные в таблицу. |
| void | <code>printFile()</code> | Выводит файл на печать. |
| <code>String[][]</code> | <code>readFile(String pathToFile)</code> | Читает содержимое CSV-файла и возвращает его как двумерный массив строк. |
| void | <code>saveFile()</code> | Сохраняет данные таблицы в текущий CSV/XML-файл. |
| void | <code>saveFileAs()</code> | Открывает диалоговое окно для выбора нового места и имени для сохранения CSV-файла. |
| private void | <code>tableFill(String[] text)</code> | Заполняет таблицу данными из двумерного массива строк. |
| private void | <code>tableFill(Document data)</code> | Заполняет таблицу данными из двумерного массива строк. |
| void | <code>writeFile(String pathToFile)</code> | Записывает содержимое таблицы в CSV-файл. |
| private void | <code>writeFile(Document data)</code> | Записывает содержимое таблицы в XML-файл. |

Methods inherited from class `java.lang.Object`

`clone`², `equals`², `finalize`², `getClass`², `hashCode`², `notify`², `notifyAll`², `toString`², `wait`², `wait`², `wait`²

Field Details

window

```
public JFrame window
```

Основное окно приложения


```
public JTable table
```

Таблица для отображения данных

| tableModel |
|------------|
|------------|

```
public DefaultTableModel tableModel
```

Модель таблицы для управления данными

currentFilePath

```
public String2 currentFilePath
```

Текущий путь к файлу

log

```
private static final org.apache.log4j.Logger log
```

Constructor Details

Processor

Processor(JFrame^o frame)

Конструктор, принимающий основное окно приложения.

Parameters:

frame - основное окно приложения JFrame

Method Details

openFile

public void openFile()

Открывает диалоговое окно для выбора CSV-файла и загружает данные в таблицу.

saveFile

public void saveFile()

Сохраняет данные таблицы в текущий CSV/XML-файл.

saveFileAs

public void saveFileAs()

Открывает диалоговое окно для выбора нового места и имени для сохранения CSV-файла.

printFile

public void printFile()

Выводит файл на печать.

exportFile

public void exportFile(String^o format)

Экспортирует файлы.

Parameters:

format - Формат файла "html" или "pdf"

getReport

private net.sf.jasperreports.engine.JasperPrint getReport(Document^o doc)

Конвертирует таблицу в объект для печати или экспорта

Parameters:

doc - объект Document содержащий xml таблицу

Returns:

объект JasperPrint для экспорта или печати или null в случае ошибки

readFile

public String^o[][] readFile(String^o pathToFile)

Читает содержимое CSV-файла и возвращает его как двумерный массив строк.

Parameters:

pathToFile - путь к файлу для чтения

Returns:

двумерный массив строк, представляющий данные таблицы, или null в случае ошибки чтения файла

writeFile

public void writeFile(String^o pathToFile)

Записывает содержимое таблицы в CSV-файл.

Parameters:

pathToFile - путь к файлу для записи

writeFile

private void writeFile(Document^o data)

Записывает содержимое таблицы в XML-файл.

Parameters:

data - объект xml документа

tableFill

private void tableFill(String^o[][] text)

Заполняет таблицу данными из двумерного массива строк.

Parameters:

text - двумерный массив строк, представляющий данные для таблицы

tableFill

private void tableFill(Document^o data)

Заполняет таблицу данными из двумерного массива строк.

Parameters:

data - объект xml документа

createTable

```
public void createTable(String[] colNames)
```

Создает таблицу с заданными именами столбцов.

Parameters:

colNames - массив строк с именами столбцов

Вывод

В результате выполнения работы были изучены правила работы с протоколированием приложения в java и получены практические навыки в программировании на этом языке.

Ссылки

https://github.com/DanyaMokhno/OOP_Labs/tree/main/com.study_oop.Laba_10