**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**

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

Выполнил: Барченков П. А.

Факультет: КТИ

Группа: №3312

Подпись преподавателя: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2024

**Содержание**

[Цель работы 3](#_Toc183897975)

[Перечень используемых типов сообщений, которые выводятся в лог-файл 3](#_Toc183897976)

[Конфигурационный файл log4j.properties 4](#_Toc183897977)

[Лог-файлы работы приложения в режимах WARN+INFO и DEBUG 4](#_Toc183897978)

[Текст программы 5](#_Toc183897979)

[Приложение 20](#_Toc183897980)

# Цель работы

Знакомство с методами протоколирования работы приложения с использованием библиотеки Log4j в языке программирования Java.

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

1. DEBUG (Отладочная информация):

* Используется для вывода подробной отладочной информации, которая полезна при разработке и отладке программы. Эти сообщения позволяют отслеживать выполнение кода на низком уровне и понимать, как работает приложение.
* Этот уровень используется для отслеживания шагов в методах, значений переменных и других технических подробностей, которые помогают в отладке.

1. INFO (Информационные сообщения):

* Используется для логирования основных событий, которые отражают нормальное функционирование приложения. Эти сообщения могут быть полезны для мониторинга работы приложения в реальном времени.
* Этот уровень используется для фиксации успешных завершений операций и других значимых, но не критичных событий.

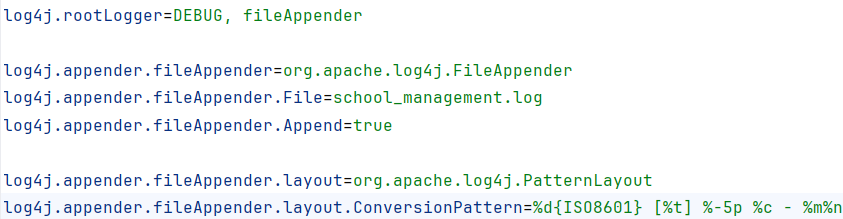
1. WARN (Предупреждения):

* Используется для сообщений, которые указывают на потенциальные проблемы, которые могут возникнуть в будущем, но не мешают продолжению работы программы. Такие события не являются фатальными.
* Этот уровень используется для предупреждения о ситуации, которая требует внимания, но не прерывает работу приложения.

1. ERROR (Сообщения об ошибках):

* Используется для логирования ошибок, которые произошли во время выполнения программы и могут нарушать её нормальную работу.
* Этот уровень применяется для записи исключений, непредвиденных ситуаций и других критических ошибок, требующих немедленного внимания.

# Конфигурационный файл log4j.properties



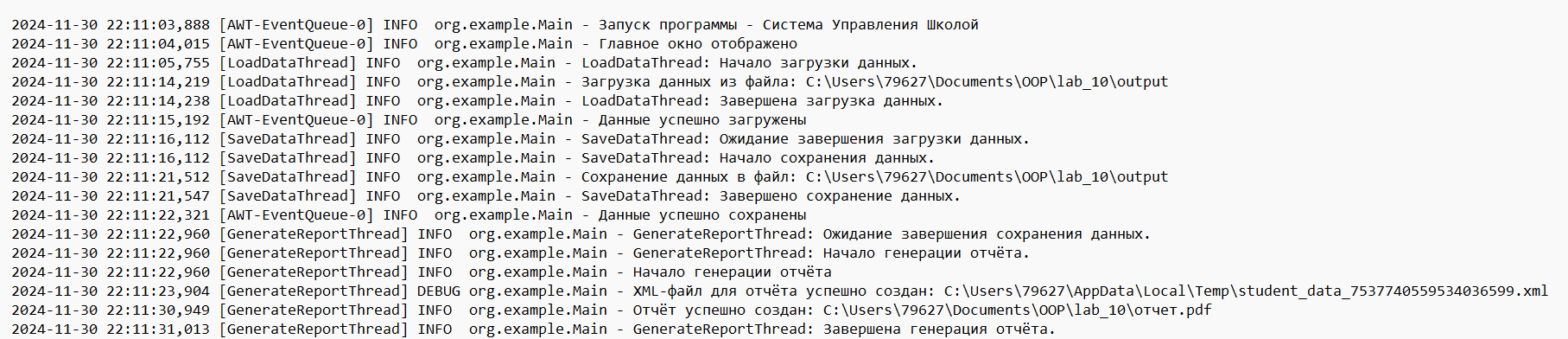
Лог-файлы работы приложения в режимах WARN+INFO и DEBUG

Рисунок 1 – Лог-файл myproject.log в режиме DEBUG

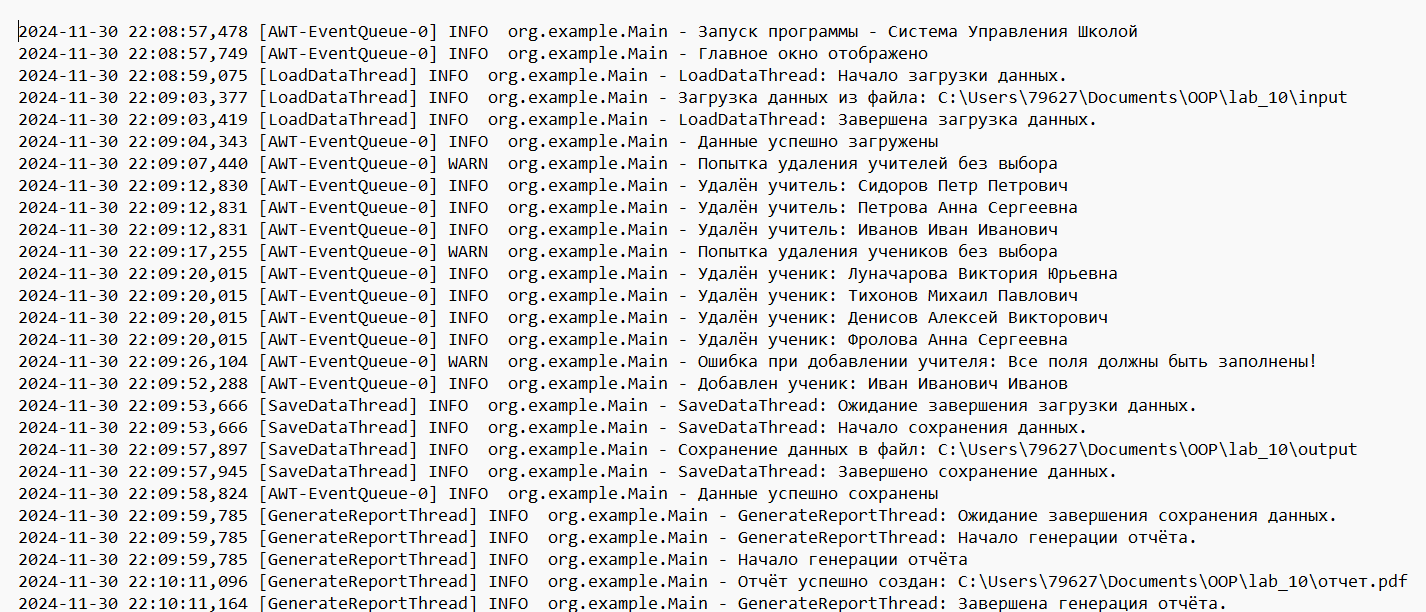


Рисунок 2 – Лог-файл myproject1.log в режиме WARN+INFO

# Текст программы

|  |
| --- |
| **package org.example;  import javax.swing.\*; import javax.swing.table.DefaultTableModel; import javax.swing.table.TableRowSorter; import javax.swing.RowFilter; import javax.swing.event.ChangeListener; import javax.swing.event.ChangeEvent; import java.awt.\*; import java.awt.event.\*; import java.io.\*; import java.util.ArrayList; import java.util.List; import java.util.concurrent.CountDownLatch;  // Импорты для работы с XML import org.w3c.dom.\*; import javax.xml.parsers.\*; import org.xml.sax.SAXException; import javax.xml.transform.\*; import javax.xml.transform.dom.DOMSource; import javax.xml.transform.stream.StreamResult;  // Импорты для JasperReports import net.sf.jasperreports.engine.\*; import net.sf.jasperreports.engine.data.JRXmlDataSource; import net.sf.jasperreports.view.JasperViewer; import java.util.HashMap;  // Импорт для логирования import org.apache.log4j.Logger; import org.apache.log4j.PropertyConfigurator;  /\*\*  \* Программа для управления данными учителей и учеников в системе управления школой.  \* Содержит функции добавления, удаления учителей и учеников, а также поиска, фильтрации,  \* загрузки и сохранения данных из/в XML-файлы и генерации отчётов.  \* @author Барченков Платон 3312  \* @version 1.0  \*/ public class Main {  private static final Logger log = Logger.getLogger(Main.class);   private JFrame frame;  private JTable teacherTable, studentTable;  private DefaultTableModel teacherTableModel, studentTableModel;  private JPanel filterPanel;  private JButton addTeacherButton, addStudentButton, deleteTeacherButton, deleteStudentButton, generateReportButton;  private JButton searchButton, resetButton, loadButton, saveButton;  private JComboBox<String> searchCriteria;  private JTextField searchField;  private JScrollPane teacherScrollPane, studentScrollPane;  private JTabbedPane tabbedPane;  private List<String[]> originalTeacherData; // Исходные данные учителей  private List<String[]> originalStudentData; // Исходные данные учеников  private TableRowSorter<DefaultTableModel> teacherSorter, studentSorter;   // CountDownLatch для синхронизации потоков  private final CountDownLatch loadLatch = new CountDownLatch(1);  private final CountDownLatch saveLatch = new CountDownLatch(1);   /\*\*  \* Метод для создания и отображения основного окна программы.  \*/  public void SchoolManagementSystem() {  log.info("Запуск программы - Система Управления Школой");   // Инициализация исходных данных  originalTeacherData = new ArrayList<>();  originalStudentData = new ArrayList<>();   // Создание главного окна программы  frame = new JFrame("Система Управления Школой");  frame.setSize(1000, 700); // Увеличиваем размер окна для двух таблиц  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Закрытие окна завершает программу  frame.setLayout(new BorderLayout()); // Устанавливаем BorderLayout для главного окна   // Создание панели инструментов с кнопками действий  JToolBar actionPanel = new JToolBar("Панель инструментов");   // Кнопки для учителей и учеников  addTeacherButton = new JButton("Добавить учителя");  addStudentButton = new JButton("Добавить ученика");  deleteTeacherButton = new JButton("Удалить учителя");  deleteStudentButton = new JButton("Удалить ученика");  generateReportButton = new JButton("Создать отчёт");   // Кнопки загрузки и сохранения данных  loadButton = new JButton("Загрузить данные");  saveButton = new JButton("Сохранить данные");   // Добавляем кнопки на панель инструментов слева  actionPanel.add(addTeacherButton);  actionPanel.add(addStudentButton);  actionPanel.add(deleteTeacherButton);  actionPanel.add(deleteStudentButton);  actionPanel.add(generateReportButton);   // Добавляем гибкое пространство, чтобы следующие кнопки были справа  actionPanel.add(Box.createHorizontalGlue());   // Добавляем кнопки загрузки и сохранения данных справа  actionPanel.add(loadButton);  actionPanel.add(saveButton);   frame.add(actionPanel, BorderLayout.NORTH); // Размещаем панель инструментов сверху   // Определяем столбцы таблицы учителей  String[] teacherColumns = {"ФИО учителя", "Предмет", "Классы"};   // Инициализация модели таблицы учителей  teacherTableModel = new DefaultTableModel(teacherColumns, 0);  teacherTable = new JTable(teacherTableModel);  teacherTable.setSelectionMode(ListSelectionModel.MULTIPLE\_INTERVAL\_SELECTION); // Разрешаем множественный выбор  teacherScrollPane = new JScrollPane(teacherTable);  teacherScrollPane.setBorder(BorderFactory.createTitledBorder("Учителя"));   // Создание сортировщика для таблицы учителей  teacherSorter = new TableRowSorter<>(teacherTableModel);  teacherTable.setRowSorter(teacherSorter);   // Определяем столбцы таблицы учеников  String[] studentColumns = {"ФИО ученика", "Класс", "Успеваемость"};   // Инициализация модели таблицы учеников  studentTableModel = new DefaultTableModel(studentColumns, 0);  studentTable = new JTable(studentTableModel);  studentTable.setSelectionMode(ListSelectionModel.MULTIPLE\_INTERVAL\_SELECTION); // Разрешаем множественный выбор  studentScrollPane = new JScrollPane(studentTable);  studentScrollPane.setBorder(BorderFactory.createTitledBorder("Ученики"));   // Создание сортировщика для таблицы учеников  studentSorter = new TableRowSorter<>(studentTableModel);  studentTable.setRowSorter(studentSorter);   // Создание вкладок для таблиц  tabbedPane = new JTabbedPane();  tabbedPane.addTab("Учителя", teacherScrollPane);  tabbedPane.addTab("Ученики", studentScrollPane);  frame.add(tabbedPane, BorderLayout.CENTER); // Размещаем вкладки в центре   // Создание компонентов для панели поиска и фильтрации данных  searchCriteria = new JComboBox<>();  searchField = new JTextField(20);  searchButton = new JButton("Поиск");  resetButton = new JButton("Сбросить");   // Панель фильтрации  filterPanel = new JPanel();  filterPanel.add(new JLabel("Критерий поиска: "));  filterPanel.add(searchCriteria);  filterPanel.add(new JLabel("Значение: "));  filterPanel.add(searchField);  filterPanel.add(searchButton);  filterPanel.add(resetButton);  frame.add(filterPanel, BorderLayout.SOUTH); // Размещаем панель фильтрации снизу   // Действие при переключении вкладок для обновления критериев поиска  tabbedPane.addChangeListener(new ChangeListener() {  @Override  public void stateChanged(ChangeEvent e) {  updateSearchCriteria();  }  });   // Инициализация критериев поиска по текущей вкладке  updateSearchCriteria();   // Добавление слушателей к кнопкам  addListeners();   // Установка начального состояния кнопок  setInitialButtonStates();   // Логирование успешного запуска  log.info("Главное окно отображено");   // Делаем главное окно видимым  frame.setVisible(true);  }   /\*\*  \* Метод для установки начального состояния кнопок.  \* Только кнопка "Загрузить данные" активна.  \*/  private void setInitialButtonStates() {  addTeacherButton.setEnabled(false);  addStudentButton.setEnabled(false);  deleteTeacherButton.setEnabled(false);  deleteStudentButton.setEnabled(false);  saveButton.setEnabled(false);  generateReportButton.setEnabled(false);  }   /\*\*  \* Метод для добавления слушателей к кнопкам  \*/  private void addListeners() {  // Слушатель для кнопки "Поиск"  searchButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  String criterion = (String) searchCriteria.getSelectedItem();  String value = searchField.getText().trim();  searchTable(criterion, value);  }  });   // Слушатель для кнопки "Сбросить"  resetButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  resetTable();  }  });   // Слушатель для кнопки "Добавить учителя"  addTeacherButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  addTeacherDialog();  }  });   // Слушатель для кнопки "Удалить учителя"  deleteTeacherButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  deleteSelectedTeachers();  }  });   // Слушатель для кнопки "Добавить ученика"  addStudentButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  addStudentDialog();  }  });   // Слушатель для кнопки "Удалить ученика"  deleteStudentButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  deleteSelectedStudents();  }  });   // Слушатель для кнопки "Загрузить данные"  loadButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  setButtonStatesDuringLoad(true);  Thread loadThread = new Thread(new LoadDataThread(Main.this, loadLatch), "LoadDataThread");  loadThread.start();  }  });   // Слушатель для кнопки "Сохранить данные"  saveButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  generateReportButton.setEnabled(false);  Thread saveThread = new Thread(new SaveDataThread(Main.this, loadLatch, saveLatch), "SaveDataThread");  saveThread.start();  }  });   // Слушатель для кнопки "Создать отчёт"  generateReportButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  Thread reportThread = new Thread(new GenerateReportThread(Main.this, saveLatch), "GenerateReportThread");  reportThread.start();  }  });  }   /\*\*  \* Метод для управления состоянием кнопок во время загрузки данных.  \*  \* @param isLoading true, если данные загружаются; false иначе.  \*/  private void setButtonStatesDuringLoad(boolean isLoading) {  addTeacherButton.setEnabled(!isLoading && loadLatch.getCount() == 0);  addStudentButton.setEnabled(!isLoading && loadLatch.getCount() == 0);  deleteTeacherButton.setEnabled(!isLoading && loadLatch.getCount() == 0);  deleteStudentButton.setEnabled(!isLoading && loadLatch.getCount() == 0);  saveButton.setEnabled(!isLoading && loadLatch.getCount() == 0);  generateReportButton.setEnabled(false);  loadButton.setEnabled(!isLoading); // Разрешаем повторную загрузку  }   /\*\*  \* Метод для управления состоянием кнопок после загрузки данных.  \*  \* @param isLoaded true, если данные успешно загружены; false иначе.  \*/  private void setButtonStatesAfterLoad(boolean isLoaded) {  if (isLoaded) {  addTeacherButton.setEnabled(true);  addStudentButton.setEnabled(true);  deleteTeacherButton.setEnabled(true);  deleteStudentButton.setEnabled(true);  saveButton.setEnabled(true);  } else {  addTeacherButton.setEnabled(false);  addStudentButton.setEnabled(false);  deleteTeacherButton.setEnabled(false);  deleteStudentButton.setEnabled(false);  saveButton.setEnabled(false);  }  // Кнопка создания отчёта остаётся отключённой до сохранения  generateReportButton.setEnabled(false);  }   /\*\*  \* Обновляет критерии поиска в зависимости от выбранной вкладки.  \*/  private void updateSearchCriteria() {  int selectedIndex = tabbedPane.getSelectedIndex();  searchCriteria.removeAllItems();   if (selectedIndex == 0) { // Учителя  searchCriteria.addItem("ФИО учителя");  searchCriteria.addItem("Предмет");  searchCriteria.addItem("Классы");  } else if (selectedIndex == 1) { // Ученики  searchCriteria.addItem("ФИО ученика");  searchCriteria.addItem("Класс ученика");  searchCriteria.addItem("Успеваемость");  }  }   /\*\*  \* Метод для фильтрации данных в таблице на основе критерия и значения поиска.  \*  \* @param criterion Критерий поиска.  \* @param value Значение для поиска.  \*/  private void searchTable(String criterion, String value) {  if (value.isEmpty()) {  JOptionPane.showMessageDialog(frame, "Поле поиска не может быть пустым.", "Ошибка", JOptionPane.ERROR\_MESSAGE);  log.warn("Попытка выполнить поиск с пустым значением");  return;  }   int selectedIndex = tabbedPane.getSelectedIndex();   if (selectedIndex == 0) { // Учителя  int columnIndex = -1;  switch (criterion) {  case "ФИО учителя":  columnIndex = 0;  break;  case "Предмет":  columnIndex = 1;  break;  case "Классы":  columnIndex = 2;  break;  }   if (columnIndex != -1) {  teacherSorter.setRowFilter(RowFilter.regexFilter("(?i)" + value, columnIndex));  log.info("Выполнен поиск учителей по критерию: " + criterion + " с значением: " + value);  }  } else if (selectedIndex == 1) { // Ученики  int columnIndex = -1;  switch (criterion) {  case "ФИО ученика":  columnIndex = 0;  break;  case "Класс ученика":  columnIndex = 1;  break;  case "Успеваемость":  columnIndex = 2;  break;  }   if (columnIndex != -1) {  studentSorter.setRowFilter(RowFilter.regexFilter("(?i)" + value, columnIndex));  log.info("Выполнен поиск учеников по критерию: " + criterion + " с значением: " + value);  }  }  }   /\*\*  \* Метод для сброса фильтров и восстановления исходных данных.  \*/  private void resetTable() {  // Сброс фильтра для учителей  teacherSorter.setRowFilter(null);  // Сброс фильтра для учеников  studentSorter.setRowFilter(null);  // Очистка поля поиска  searchField.setText("");  log.info("Фильтры поиска сброшены");  }   /\*\*  \* Метод для загрузки данных из файла, выбранного пользователем.  \*/  private void loadDataFromFile() {  JFileChooser fileChooser = new JFileChooser();  fileChooser.setDialogTitle("Выберите XML-файл для загрузки данных");  int userSelection = fileChooser.showOpenDialog(frame);   if (userSelection == JFileChooser.APPROVE\_OPTION) {  File xmlFile = fileChooser.getSelectedFile();  try {  log.info("Загрузка данных из файла: " + xmlFile.getAbsolutePath());   DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  DocumentBuilder builder = factory.newDocumentBuilder();  Document doc = builder.parse(xmlFile);   // Нормализация документа  doc.getDocumentElement().normalize();   // Очистка текущих данных в таблицах и исходных списках  SwingUtilities.invokeLater(() -> {  teacherTableModel.setRowCount(0);  studentTableModel.setRowCount(0);  originalTeacherData.clear();  originalStudentData.clear();  });   // Загрузка учителей  NodeList teacherList = doc.getElementsByTagName("teacher");  for (int i = 0; i < teacherList.getLength(); i++) {  Element teacherElement = (Element) teacherList.item(i);  String name = teacherElement.getAttribute("name");  String subject = teacherElement.getAttribute("subject");  String classes = teacherElement.getAttribute("classes");   String[] teacher = {name, subject, classes};  SwingUtilities.invokeLater(() -> {  teacherTableModel.addRow(teacher);  originalTeacherData.add(teacher);  });  }   // Загрузка учеников  NodeList studentList = doc.getElementsByTagName("student");  for (int i = 0; i < studentList.getLength(); i++) {  Element studentElement = (Element) studentList.item(i);  String name = studentElement.getAttribute("name");  String studentClass = studentElement.getAttribute("class");  String performance = studentElement.getAttribute("performance");   String[] student = {name, studentClass, performance};  SwingUtilities.invokeLater(() -> {  studentTableModel.addRow(student);  originalStudentData.add(student);  });  }   SwingUtilities.invokeLater(() -> {  JOptionPane.showMessageDialog(frame, "Данные успешно загружены из XML-файла.", "Успех", JOptionPane.INFORMATION\_MESSAGE);  setButtonStatesAfterLoad(true);  log.info("Данные успешно загружены");  });   } catch (ParserConfigurationException | SAXException | IOException e) {  SwingUtilities.invokeLater(() -> {  JOptionPane.showMessageDialog(frame, "Ошибка при загрузке данных: " + e.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);  });  log.error("Ошибка при загрузке данных: " + e.getMessage(), e);  }  }  }   /\*\*  \* Метод для сохранения данных в файл, выбранный пользователем.  \*/  private void saveDataToFile() {  JFileChooser fileChooser = new JFileChooser();  fileChooser.setDialogTitle("Сохраните данные в XML-файл");  int userSelection = fileChooser.showSaveDialog(frame);   if (userSelection == JFileChooser.APPROVE\_OPTION) {  File xmlFile = fileChooser.getSelectedFile();  try {  log.info("Сохранение данных в файл: " + xmlFile.getAbsolutePath());   // Создание фабрики и построителя документов  DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();  DocumentBuilder docBuilder = docFactory.newDocumentBuilder();   // Создание нового документа  Document doc = docBuilder.newDocument();   // Создание корневого элемента <school>  Element rootElement = doc.createElement("school");  doc.appendChild(rootElement);   // Создание элемента <teachers>  Element teachersElement = doc.createElement("teachers");  rootElement.appendChild(teachersElement);   // Добавление каждого учителя как элемента <teacher>  for (String[] teacher : originalTeacherData) {  Element teacherElement = doc.createElement("teacher");  teacherElement.setAttribute("name", teacher[0]);  teacherElement.setAttribute("subject", teacher[1]);  teacherElement.setAttribute("classes", teacher[2]);  teachersElement.appendChild(teacherElement);  }   // Создание элемента <students>  Element studentsElement = doc.createElement("students");  rootElement.appendChild(studentsElement);   // Добавление каждого ученика как элемента <student>  for (String[] student : originalStudentData) {  Element studentElement = doc.createElement("student");  studentElement.setAttribute("name", student[0]);  studentElement.setAttribute("class", student[1]);  studentElement.setAttribute("performance", student[2]);  studentsElement.appendChild(studentElement);  }   // Создание преобразователя и запись документа в файл  TransformerFactory transformerFactory = TransformerFactory.newInstance();  Transformer transformer = transformerFactory.newTransformer();  // Для красивого форматирования XML  transformer.setOutputProperty(OutputKeys.INDENT, "yes");  transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");  DOMSource source = new DOMSource(doc);   StreamResult result = new StreamResult(xmlFile);  transformer.transform(source, result);   SwingUtilities.invokeLater(() -> {  JOptionPane.showMessageDialog(frame, "Данные успешно сохранены в XML-файл.", "Успех", JOptionPane.INFORMATION\_MESSAGE);  generateReportButton.setEnabled(true);  log.info("Данные успешно сохранены");  });   } catch (ParserConfigurationException | TransformerException e) {  SwingUtilities.invokeLater(() -> {  JOptionPane.showMessageDialog(frame, "Ошибка при сохранении данных: " + e.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);  });  log.error("Ошибка при сохранении данных: " + e.getMessage(), e);  }  }  }   /\*\*  \* Метод для отображения диалогового окна добавления учителя  \*/  private void addTeacherDialog() {  JPanel panel = new JPanel(new GridLayout(3, 2));  JTextField nameField = new JTextField(20);  JTextField subjectField = new JTextField(20);  JTextField classesField = new JTextField(20);   panel.add(new JLabel("ФИО учителя: "));  panel.add(nameField);  panel.add(new JLabel("Предмет: "));  panel.add(subjectField);  panel.add(new JLabel("Классы: "));  panel.add(classesField);   int result = JOptionPane.showConfirmDialog(frame, panel, "Добавить учителя", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);   if (result == JOptionPane.OK\_OPTION) {  try {  String name = nameField.getText();  String subject = subjectField.getText();  String classes = classesField.getText();   validateFields(name, subject, classes);   // Добавление в таблицу и исходные данные  String[] newTeacher = {name, subject, classes};  teacherTableModel.addRow(newTeacher);  originalTeacherData.add(newTeacher);   JOptionPane.showMessageDialog(frame, "Учитель добавлен!", "Добавление", JOptionPane.INFORMATION\_MESSAGE);  log.info("Добавлен учитель: " + name);   } catch (InvalidFieldException ex) {  JOptionPane.showMessageDialog(frame, ex.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);  log.warn("Ошибка при добавлении учителя: " + ex.getMessage());  }  }  }   /\*\*  \* Метод для отображения диалогового окна добавления ученика  \*/  private void addStudentDialog() {  JPanel panel = new JPanel(new GridLayout(3, 2));  JTextField nameField = new JTextField(20);  JTextField classField = new JTextField(20);  JTextField performanceField = new JTextField(20);   panel.add(new JLabel("ФИО ученика: "));  panel.add(nameField);  panel.add(new JLabel("Класс: "));  panel.add(classField);  panel.add(new JLabel("Успеваемость: "));  panel.add(performanceField);   int result = JOptionPane.showConfirmDialog(frame, panel, "Добавить ученика", JOptionPane.OK\_CANCEL\_OPTION, JOptionPane.PLAIN\_MESSAGE);   if (result == JOptionPane.OK\_OPTION) {  try {  String name = nameField.getText();  String className = classField.getText();  String performance = performanceField.getText();   validateFields(name, className, performance);   // Добавление в таблицу и исходные данные  String[] newStudent = {name, className, performance};  studentTableModel.addRow(newStudent);  originalStudentData.add(newStudent);   JOptionPane.showMessageDialog(frame, "Ученик добавлен!", "Добавление", JOptionPane.INFORMATION\_MESSAGE);  log.info("Добавлен ученик: " + name);   } catch (InvalidFieldException ex) {  JOptionPane.showMessageDialog(frame, ex.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);  log.warn("Ошибка при добавлении ученика: " + ex.getMessage());  }  }  }   /\*\*  \* Метод для проверки корректности введенных данных  \*  \* @param name ФИО учителя или ученика  \* @param field1 Предмет учителя или класс ученика  \* @param field2 Классы учителя или успеваемость ученика  \* @throws InvalidFieldException если хотя бы одно из полей пустое  \*/  private void validateFields(String name, String field1, String field2) throws InvalidFieldException {  if (name.isEmpty() || field1.isEmpty() || field2.isEmpty()) {  throw new InvalidFieldException("Все поля должны быть заполнены!");  }  // Дополнительная валидация может быть добавлена здесь  }   /\*\*  \* Метод для удаления выбранных учителей с подтверждением.  \*/  private void deleteSelectedTeachers() {  int[] selectedRows = teacherTable.getSelectedRows();  if (selectedRows.length == 0) {  JOptionPane.showMessageDialog(frame, "Пожалуйста, выберите учителей для удаления.", "Ошибка удаления", JOptionPane.ERROR\_MESSAGE);  log.warn("Попытка удаления учителей без выбора");  return;  }   // Получение имен выбранных учителей  StringBuilder names = new StringBuilder();  for (int row : selectedRows) {  int modelRow = teacherTable.convertRowIndexToModel(row);  String teacherName = (String) teacherTableModel.getValueAt(modelRow, 0);  names.append(teacherName).append("\n");  }   // Определяем тексты кнопок на русском  String[] options = {"Да", "Нет"};   // Показываем диалог подтверждения с русскими кнопками  int confirm = JOptionPane.showOptionDialog(  frame,  "Вы уверены, что хотите удалить следующих учителей?\n" + names.toString(),  "Подтверждение удаления",  JOptionPane.YES\_NO\_OPTION,  JOptionPane.WARNING\_MESSAGE,  null,  options,  options[0]  );   if (confirm == JOptionPane.YES\_OPTION) {  // Сортировка выбранных строк по убыванию индексов  int[] sortedRows = selectedRows.clone();  java.util.Arrays.sort(sortedRows);  for (int i = sortedRows.length - 1; i >= 0; i--) {  int modelRow = teacherTable.convertRowIndexToModel(sortedRows[i]);  String removedTeacher = (String) teacherTableModel.getValueAt(modelRow, 0);  teacherTableModel.removeRow(modelRow);  originalTeacherData.remove(modelRow);  log.info("Удалён учитель: " + removedTeacher);  }  JOptionPane.showMessageDialog(frame, "Учителя удалены.", "Удаление", JOptionPane.INFORMATION\_MESSAGE);  } else {  log.info("Удаление учителей отменено пользователем");  }  }   /\*\*  \* Метод для удаления выбранных учеников с подтверждением.  \*/  private void deleteSelectedStudents() {  int[] selectedRows = studentTable.getSelectedRows();  if (selectedRows.length == 0) {  JOptionPane.showMessageDialog(frame, "Пожалуйста, выберите учеников для удаления.", "Ошибка удаления", JOptionPane.ERROR\_MESSAGE);  log.warn("Попытка удаления учеников без выбора");  return;  }   // Получение имен выбранных учеников  StringBuilder names = new StringBuilder();  for (int row : selectedRows) {  int modelRow = studentTable.convertRowIndexToModel(row);  String studentName = (String) studentTableModel.getValueAt(modelRow, 0);  names.append(studentName).append("\n");  }   // Определяем тексты кнопок на русском  String[] options = {"Да", "Нет"};   // Показываем диалог подтверждения с русскими кнопками  int confirm = JOptionPane.showOptionDialog(  frame,  "Вы уверены, что хотите удалить следующих учеников?\n" + names.toString(),  "Подтверждение удаления",  JOptionPane.YES\_NO\_OPTION,  JOptionPane.WARNING\_MESSAGE,  null,  options,  options[0]  );   if (confirm == JOptionPane.YES\_OPTION) {  // Сортировка выбранных строк по убыванию индексов  int[] sortedRows = selectedRows.clone();  java.util.Arrays.sort(sortedRows);  for (int i = sortedRows.length - 1; i >= 0; i--) {  int modelRow = studentTable.convertRowIndexToModel(sortedRows[i]);  String removedStudent = (String) studentTableModel.getValueAt(modelRow, 0);  studentTableModel.removeRow(modelRow);  originalStudentData.remove(modelRow);  log.info("Удалён ученик: " + removedStudent);  }  JOptionPane.showMessageDialog(frame, "Ученики удалены.", "Удаление", JOptionPane.INFORMATION\_MESSAGE);  } else {  log.info("Удаление учеников отменено пользователем");  }  }   /\*\*  \* Метод запуска всех потоков  \*/  private void startThreads() {  startLoad();  startEditAndSave();  startReport();  }   /\*\*  \* Первый поток для загрузки данных из XML  \*/  private void startLoad() {  Thread thread1 = new Thread(new LoadDataThread(Main.this, loadLatch), "LoadDataThread");  thread1.start();  }   /\*\*  \* Второй поток для редактирования и сохранения в XML  \*/  private void startEditAndSave() {  Thread thread2 = new Thread(new SaveDataThread(Main.this, loadLatch, saveLatch), "SaveDataThread");  thread2.start();  }   /\*\*  \* Третий поток для формирования отчета в HTML/PDF  \*/  private void startReport() {  Thread thread3 = new Thread(new GenerateReportThread(Main.this, saveLatch), "GenerateReportThread");  thread3.start();  }   /\*\*  \* Метод для генерации отчёта.  \*/  public void generateReport() {  log.info("Начало генерации отчёта");   try {  // Запрос формата отчёта у пользователя (PDF или HTML)  String[] options = {"PDF", "HTML"};  int choice = JOptionPane.showOptionDialog(  frame,  "Выберите формат отчёта для сохранения:",  "Выбор формата отчёта",  JOptionPane.DEFAULT\_OPTION,  JOptionPane.INFORMATION\_MESSAGE,  null,  options,  options[0]  );   if (choice == JOptionPane.CLOSED\_OPTION) {  log.info("Генерация отчёта отменена пользователем");  return; // Пользователь закрыл диалог без выбора  }   String selectedFormat = options[choice];   // Создание XML-документа с данными для отчёта  String xmlDataPath = createDataXML();   if (xmlDataPath == null) {  JOptionPane.showMessageDialog(frame, "Не удалось создать XML-файл с данными для отчёта.", "Ошибка", JOptionPane.ERROR\_MESSAGE);  log.error("Не удалось создать XML-файл для отчёта");  return;  }   // Компиляция JasperReport шаблона  String reportPath = "lab\_10.jrxml"; // Убедитесь, что путь и имя файла верны  File reportFile = new File(reportPath);  if (!reportFile.exists()) {  JOptionPane.showMessageDialog(frame, "Файл шаблона отчёта не найден: " + reportPath, "Ошибка", JOptionPane.ERROR\_MESSAGE);  log.error("Файл шаблона отчёта не найден: " + reportPath);  return;  }  JasperReport jasperReport = JasperCompileManager.compileReport(reportPath);   // Создание источника данных из XML  JRXmlDataSource xmlDataSource = new JRXmlDataSource(xmlDataPath, "/school/students/student");   // Заполнение отчёта данными  JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, new HashMap<>(), xmlDataSource);   // Настройка JFileChooser для выбора места сохранения отчёта  JFileChooser fileChooser = new JFileChooser();  fileChooser.setDialogTitle("Сохранить отчёт");  if ("PDF".equalsIgnoreCase(selectedFormat)) {  fileChooser.setFileFilter(new javax.swing.filechooser.FileNameExtensionFilter("PDF файлы", "pdf"));  } else {  fileChooser.setFileFilter(new javax.swing.filechooser.FileNameExtensionFilter("HTML файлы", "html"));  }  fileChooser.setCurrentDirectory(new File(System.getProperty("user.home")));   int userSelection = fileChooser.showSaveDialog(frame);   if (userSelection == JFileChooser.APPROVE\_OPTION) {  File saveFile = fileChooser.getSelectedFile();  String filePath = saveFile.getAbsolutePath();   // Добавляем расширение, если оно отсутствует  if (!filePath.toLowerCase().endsWith("." + selectedFormat.toLowerCase())) {  filePath += "." + selectedFormat.toLowerCase();  }   if ("PDF".equalsIgnoreCase(selectedFormat)) {  JasperExportManager.exportReportToPdfFile(jasperPrint, filePath);  } else if ("HTML".equalsIgnoreCase(selectedFormat)) {  JasperExportManager.exportReportToHtmlFile(jasperPrint, filePath);  }   JOptionPane.showMessageDialog(frame, "Отчёт успешно сохранён: " + filePath, "Успех", JOptionPane.INFORMATION\_MESSAGE);  log.info("Отчёт успешно создан: " + filePath);   // Отображение отчёта  JasperViewer.viewReport(jasperPrint, false);  } else {  log.info("Сохранение отчёта отменено пользователем");  }   } catch (JRException e) {  log.error("Ошибка при генерации отчёта: " + e.getMessage(), e);  JOptionPane.showMessageDialog(frame, "Ошибка при генерации отчёта: " + e.getMessage(), "Ошибка", JOptionPane.ERROR\_MESSAGE);  }  }   /\*\*  \* Метод для создания XML-файла с данными для отчёта.  \*  \* @return Путь к созданному XML-файлу или null в случае ошибки.  \*/  private String createDataXML() {  try {  // Создание фабрики и построителя документов  DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();  DocumentBuilder docBuilder = docFactory.newDocumentBuilder();   // Создание нового документа  Document doc = docBuilder.newDocument();   // Создание корневого элемента <school>  Element rootElement = doc.createElement("school");  doc.appendChild(rootElement);   // Создание элемента <students>  Element studentsElement = doc.createElement("students");  rootElement.appendChild(studentsElement);   // Добавление каждого ученика как элемента <student>  for (String[] student : originalStudentData) {  Element studentElement = doc.createElement("student");  studentElement.setAttribute("name", student[0]);  studentElement.setAttribute("class", student[1]);  studentElement.setAttribute("performance", student[2]);  studentsElement.appendChild(studentElement);  }   // Создание временного файла для данных отчёта  File tempFile = File.createTempFile("student\_data\_", ".xml");  tempFile.deleteOnExit(); // Удаление файла при завершении программы   // Запись документа в файл  TransformerFactory transformerFactory = TransformerFactory.newInstance();  Transformer transformer = transformerFactory.newTransformer();  // Для красивого форматирования XML  transformer.setOutputProperty(OutputKeys.INDENT, "yes");  transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");  DOMSource source = new DOMSource(doc);  StreamResult result = new StreamResult(tempFile);  transformer.transform(source, result);   log.debug("XML-файл для отчёта успешно создан: " + tempFile.getAbsolutePath());   return tempFile.getAbsolutePath();   } catch (ParserConfigurationException | TransformerException | IOException e) {  log.error("Ошибка при создании XML-файла для отчёта: " + e.getMessage(), e);  return null;  }  }   /\*\*  \* Основной метод запуска приложения.  \*  \* @param args аргументы командной строки (не используются)  \*/  public static void main(String[] args) {  // Настройка Log4j из файла конфигурации  PropertyConfigurator.configure(Main.class.getClassLoader().getResource("log4j.properties"));   // Запуск интерфейса в потоке обработки событий Swing  SwingUtilities.invokeLater(() -> {  new Main().SchoolManagementSystem();  });  }   // Внутренние классы исключений   /\*\*  \* Исключение, выбрасываемое при неверном вводе данных.  \*/  class InvalidFieldException extends Exception {  public InvalidFieldException(String message) {  super(message);  }  }   /\*\*  \* Исключение, выбрасываемое если строка таблицы не выбрана.  \*/  class RowNotSelectedException extends Exception {  public RowNotSelectedException(String message) {  super(message);  }  }   /\*\*  \* Исключение, выбрасываемое если поле поиска пустое.  \*/  class EmptySearchFieldException extends Exception {  public EmptySearchFieldException(String message) {  super(message);  }  }   /\*\*  \* Класс для загрузки данных из XML-файла.  \*/  class LoadDataThread implements Runnable {  private Main mainApp;  private CountDownLatch latch;   public LoadDataThread(Main mainApp, CountDownLatch latch) {  this.mainApp = mainApp;  this.latch = latch;  }   @Override  public void run() {  log.info(Thread.currentThread().getName() + ": Начало загрузки данных.");  mainApp.loadDataFromFile();  log.info(Thread.currentThread().getName() + ": Завершена загрузка данных.");  latch.countDown(); // Уведомляем следующий поток о завершении загрузки  }  }   /\*\*  \* Класс для сохранения данных в XML-файл.  \*/  class SaveDataThread implements Runnable {  private Main mainApp;  private CountDownLatch loadLatch;  private CountDownLatch saveLatch;   public SaveDataThread(Main mainApp, CountDownLatch loadLatch, CountDownLatch saveLatch) {  this.mainApp = mainApp;  this.loadLatch = loadLatch;  this.saveLatch = saveLatch;  }   @Override  public void run() {  try {  log.info(Thread.currentThread().getName() + ": Ожидание завершения загрузки данных.");  loadLatch.await(); // Ждем завершения загрузки данных  log.info(Thread.currentThread().getName() + ": Начало сохранения данных.");   mainApp.saveDataToFile();   log.info(Thread.currentThread().getName() + ": Завершено сохранение данных.");  saveLatch.countDown(); // Уведомляем следующий поток о завершении сохранения  } catch (InterruptedException e) {  SwingUtilities.invokeLater(() -> {  JOptionPane.showMessageDialog(frame, "Поток сохранения данных был прерван.", "Ошибка", JOptionPane.ERROR\_MESSAGE);  });  log.error("Поток сохранения данных был прерван.", e);  }  }  }   /\*\*  \* Класс для генерации отчёта.  \*/  class GenerateReportThread implements Runnable {  private Main mainApp;  private CountDownLatch latch;   public GenerateReportThread(Main mainApp, CountDownLatch latch) {  this.mainApp = mainApp;  this.latch = latch;  }   @Override  public void run() {  try {  log.info(Thread.currentThread().getName() + ": Ожидание завершения сохранения данных.");  latch.await(); // Ждем завершения сохранения данных  log.info(Thread.currentThread().getName() + ": Начало генерации отчёта.");   mainApp.generateReport();   log.info(Thread.currentThread().getName() + ": Завершена генерация отчёта.");  } catch (InterruptedException e) {  SwingUtilities.invokeLater(() -> {  JOptionPane.showMessageDialog(frame, "Поток генерации отчёта был прерван.", "Ошибка", JOptionPane.ERROR\_MESSAGE);  });  log.error("Поток генерации отчёта был прерван.", e);  }  }  } }** |

# Приложение

Репозиторий: https://github.com/PlatonBarchenkov/OOP\_lab\_10.git