**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 № 4**

**«Обработка исключений»**

**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](#_Toc180854688)

[Перечень ситуаций, которые контролируются с помощью исключений 3](#_Toc180854689)

[Пример работы обработчиков ситуаций 5](#_Toc180854690)

[Текст программы 7](#_Toc180854691)

[Приложение 15](#_Toc180854692)

# Цель работы

Знакомство с механизмом обработки исключений в языке Java.

# Перечень ситуаций, которые контролируются с помощью исключений

* Пустые поля при добавлении учителя:

Описание ситуации: Пользователь пытается добавить нового учителя, не заполнив одно или несколько обязательных полей (ФИО учителя, предмет, классы). Тип исключения: InvalidInputException (пользовательское исключение). Действие: Генерируется исключение InvalidInputException с сообщением о необходимости заполнения всех обязательных полей. Это предотвращает добавление некорректных записей и информирует пользователя о необходимости ввода всех данных.

* Пустые поля при добавлении ученика:

Описание ситуации: Пользователь пытается добавить нового ученика, не заполнив одно или несколько обязательных полей (ФИО ученика, класс, успеваемость). Тип исключения: InvalidInputException (пользовательское исключение). Действие: Генерируется исключение InvalidInputException с сообщением о необходимости заполнения всех обязательных полей. Это обеспечивает целостность данных и информирует пользователя о необходимости корректного ввода информации.

* Попытка удаления учителя или ученика без выбора записи:

Описание ситуации: Пользователь пытается удалить учителя или ученика, не выбрав соответствующую строку в таблице. Тип исключения: InvalidInputException (пользовательское исключение). Действие: Генерируется исключение InvalidInputException с сообщением о необходимости выбора записи для удаления. Это предотвращает случайное удаление и информирует пользователя о необходимости выбора записи.

* Пустое поле поиска:

Описание ситуации: Пользователь пытается выполнить поиск, не введя текст в поле поиска. Тип исключения: EmptySearchException (пользовательское исключение). Действие: Генерируется исключение EmptySearchException с сообщением о необходимости ввода текста для поиска. Это предотвращает выполнение поиска без запроса и предупреждает пользователя об ошибке.

* Отсутствие значения в поле поиска:

Описание ситуации: Поле поиска содержит значение null, например, если пользователь отменил ввод. Тип исключения: NullPointerException (стандартное исключение). Действие: Генерируется стандартное исключение NullPointerException с сообщением о необходимости ввода текста для поиска. Это гарантирует, что поиск не будет выполнен, если значение отсутствует, и отображает пользователю сообщение о необходимости ввода текста.

* Ошибки при загрузке данных из файла:

Описание ситуации: При попытке загрузить данные из файла возникает ошибка ввода-вывода, например, файл не найден или поврежден. Тип исключения: DataLoadException (пользовательское исключение). Действие: Генерируется исключение DataLoadException с подробным сообщением об ошибке. Это позволяет корректно обработать проблемы при загрузке данных и информировать пользователя о возникших проблемах.

* Ошибки при сохранении данных в файл:

Описание ситуации: При попытке сохранить данные в файл возникает ошибка ввода-вывода, например, недостаточно прав доступа или место на диске закончилось. Тип исключения: DataSaveException (пользовательское исключение). Действие: Генерируется исключение DataSaveException с подробным сообщением об ошибке. Это предотвращает потерю данных и информирует пользователя о проблемах с сохранением.

# Пример работы обработчиков ситуаций

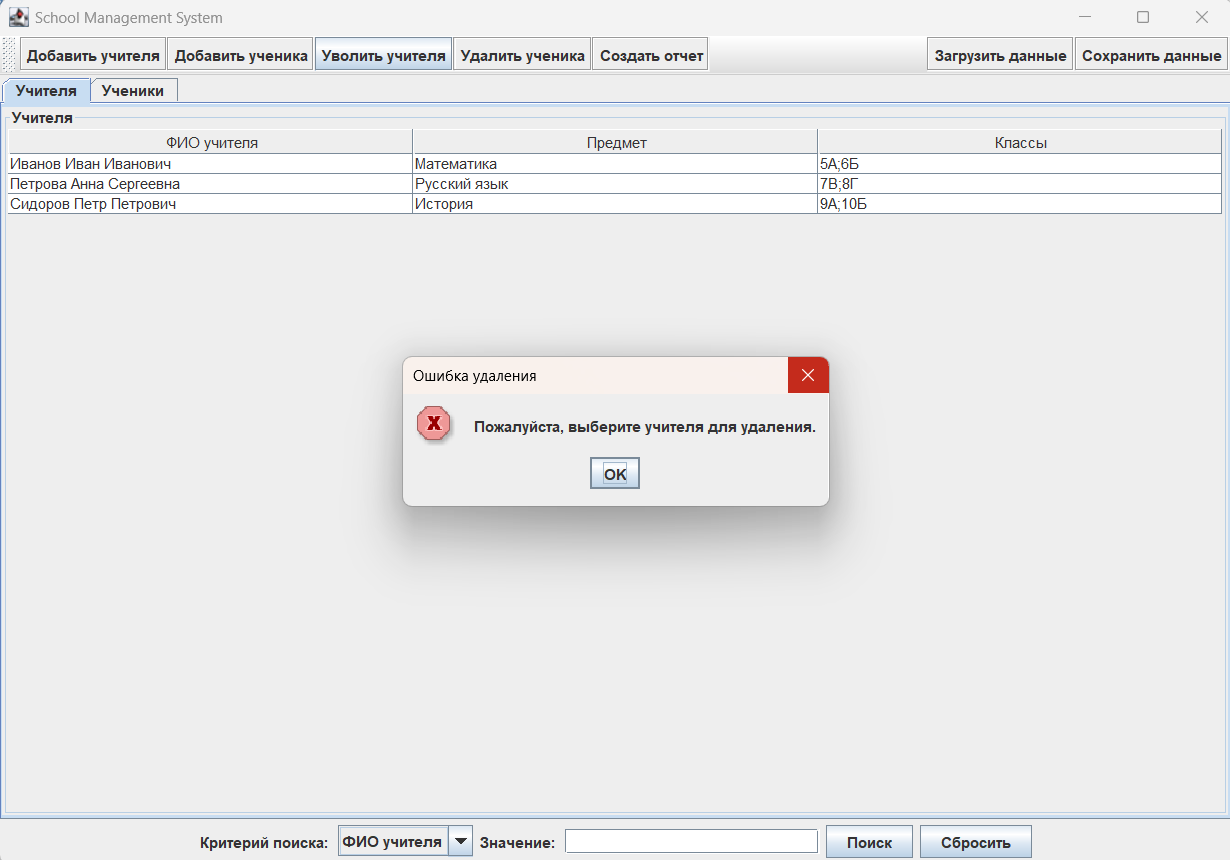
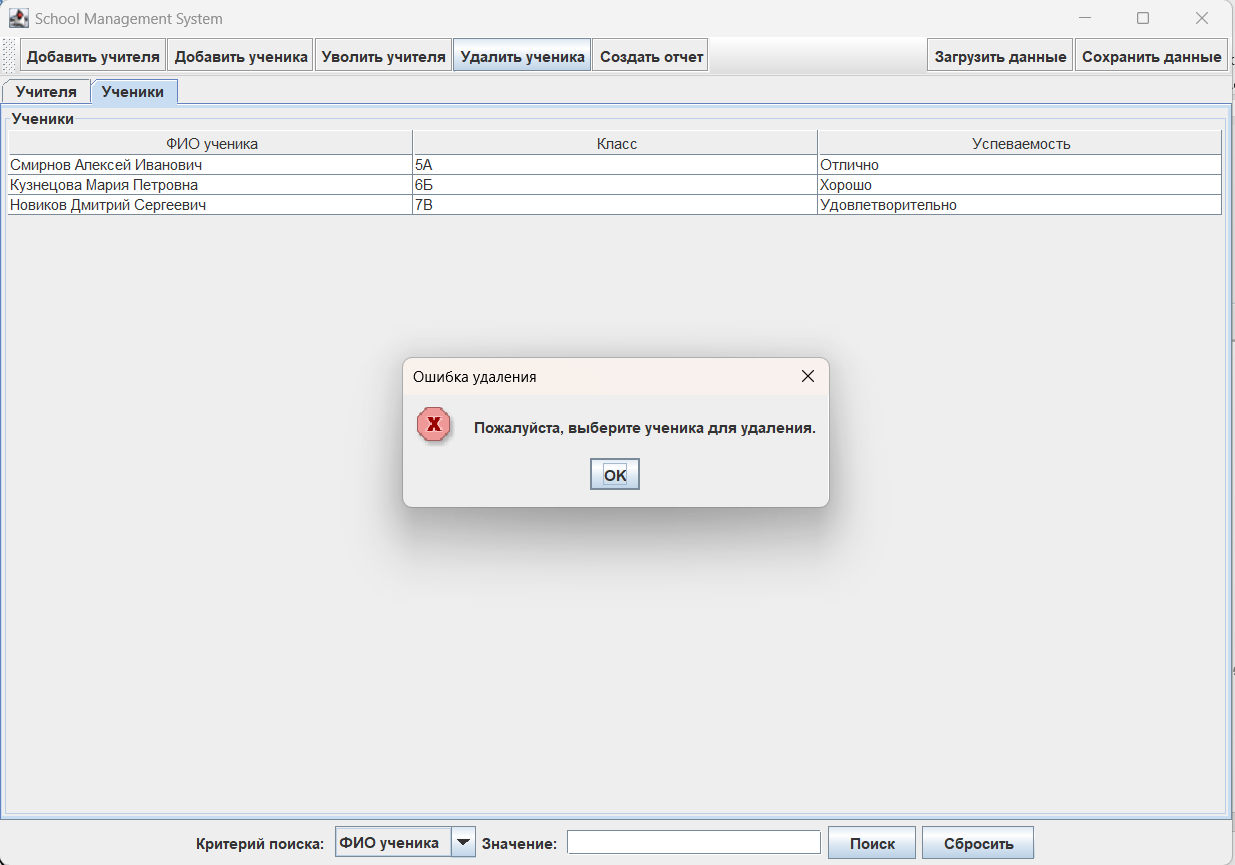


Рисунок 1 – не выбран учитель для увольнения

Рисунок 2 – не выбран ученик для удаления

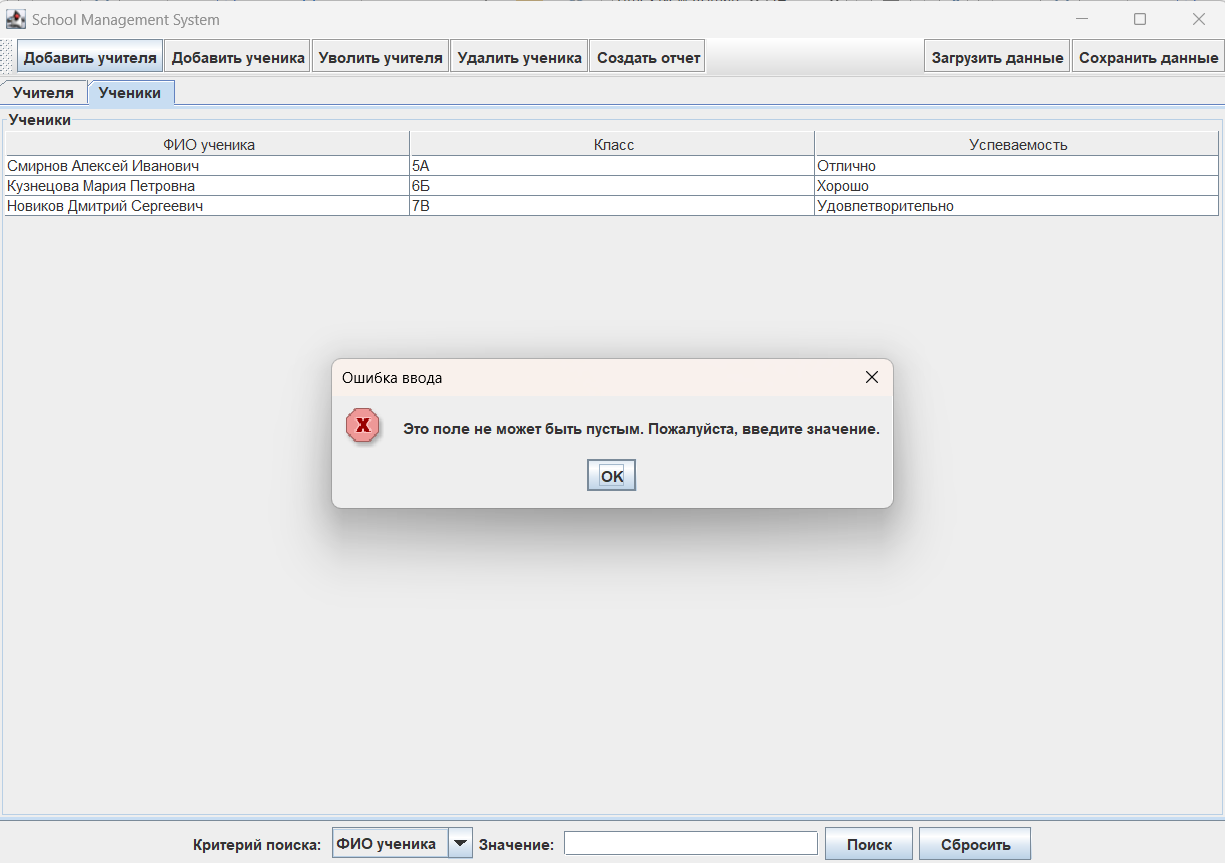


Рисунок 3 – не введено ФИО учителя

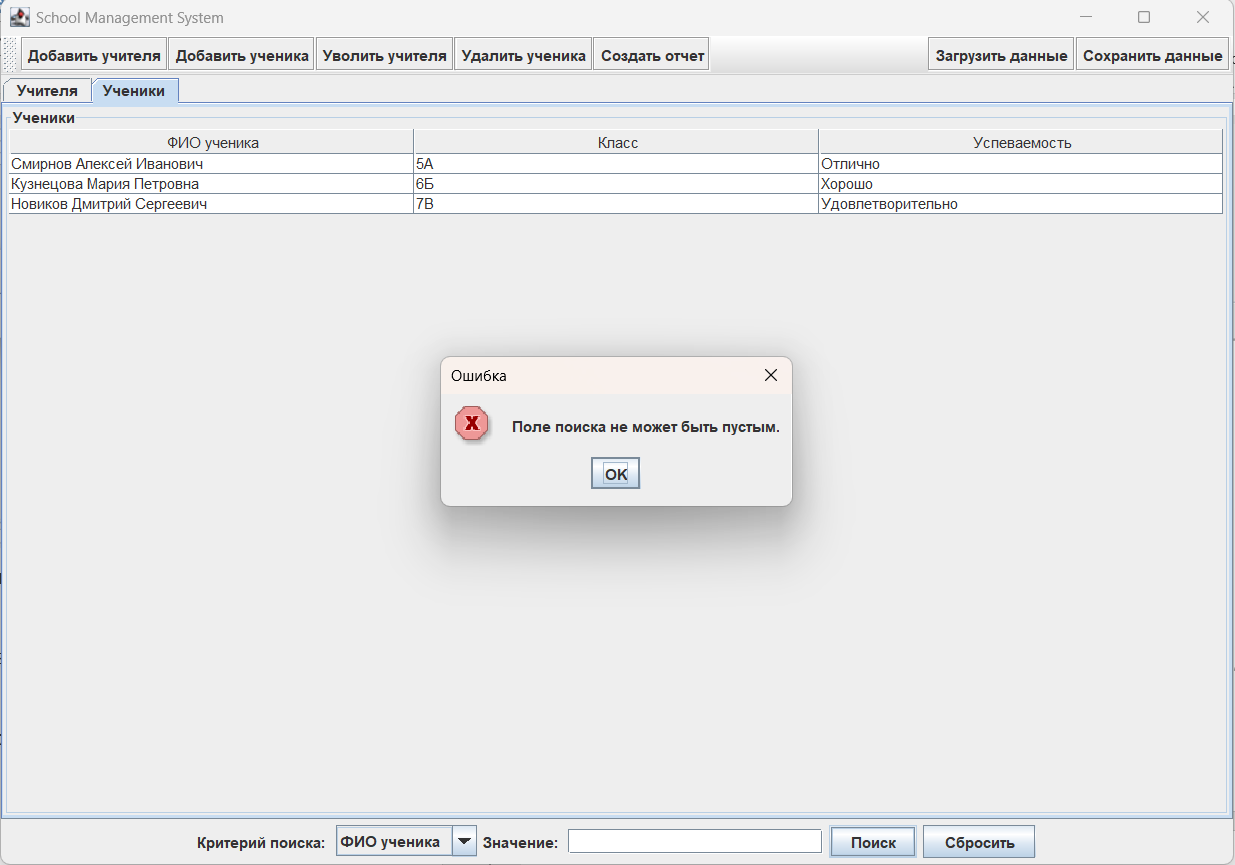


Рисунок 4 – не введено значение для поиска

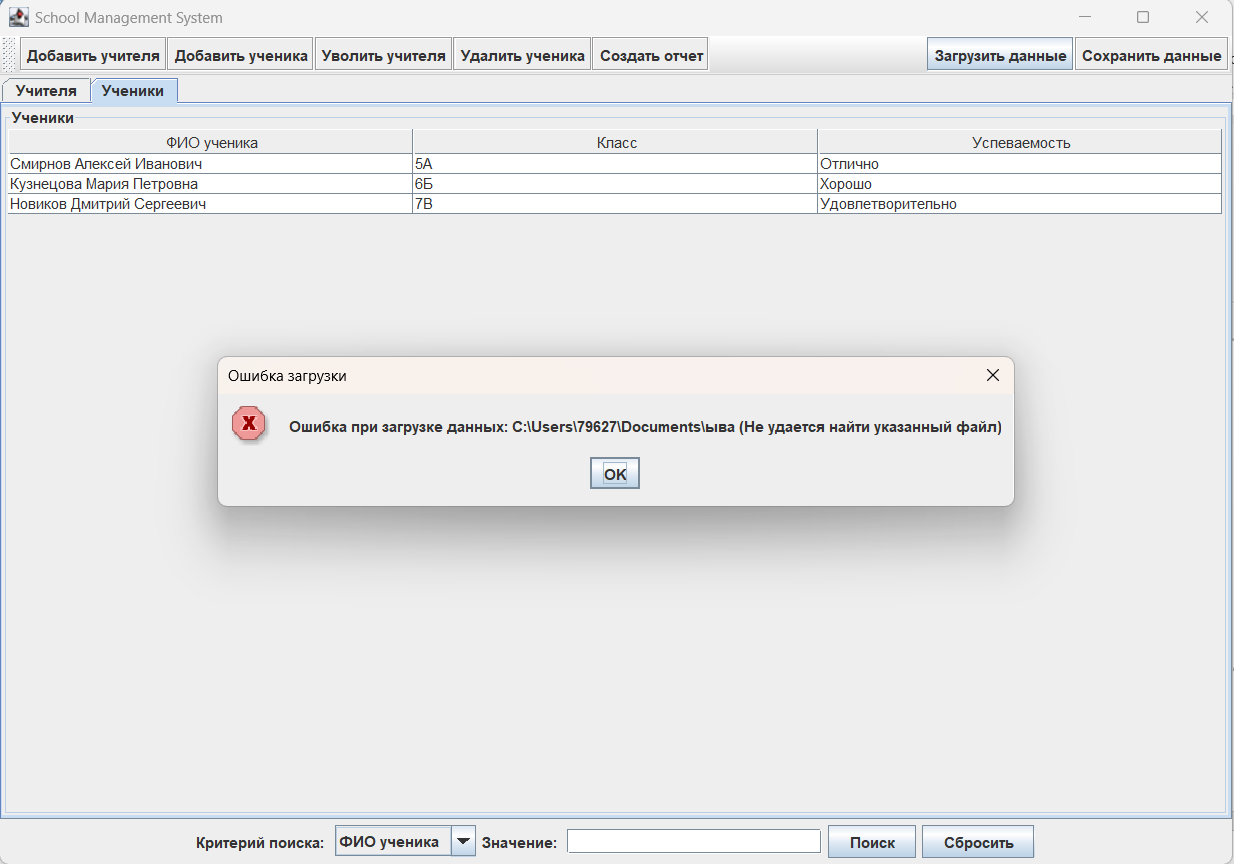


Рисунок 5 – введено не верное название файла

# Текст программы

|  |
| --- |
| import javax.swing.\*; import javax.swing.table.DefaultTableModel; import javax.swing.table.TableRowSorter; import javax.swing.event.ChangeEvent; import javax.swing.event.ChangeListener; import java.awt.\*; import java.awt.event.\*; import java.io.\*; import java.util.ArrayList; import java.util.List;  */\*\*  \* Исключение, выбрасываемое при неверном вводе данных.  \*/* class InvalidInputException extends Exception {  public InvalidInputException(String message) {  super(message);  } }  */\*\*  \* Исключение, выбрасываемое при ошибках загрузки данных из файла.  \*/* class DataLoadException extends Exception {  public DataLoadException(String message) {  super(message);  } }  */\*\*  \* Исключение, выбрасываемое при ошибках сохранения данных в файл.  \*/* class DataSaveException extends Exception {  public DataSaveException(String message) {  super(message);  } }  */\*\*  \* 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 текстовый файл.  \*  \* @autor Барченков Платон 3312  \* @version 1.4  \*/* public class Main {  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;   */\*\*  \* Метод для создания и отображения основного окна программы.  \*/* public void SchoolManagementSystem() {  *// Инициализация исходных данных* originalTeacherData = new ArrayList<>();  originalStudentData = new ArrayList<>();   *// Создание главного окна программы* frame = new JFrame("School Management System");  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("Toolbar");   *// Кнопки для учителей и учеников* addTeacherButton = new JButton("Добавить учителя");  addStudentButton = new JButton("Добавить ученика");  deleteTeacherButton = new JButton("Уволить учителя");  deleteStudentButton = new JButton("Удалить ученика");  generateReportButton = new JButton("Создать отчет");   *// 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"};  *// Исходные данные для таблицы учителей* String[][] initialTeachers = {  {"Иванов Иван Иванович", "Математика", "5А;6Б"},  {"Петрова Анна Сергеевна", "Русский язык", "7В;8Г"},  {"Сидоров Петр Петрович", "История", "9А;10Б"}  };  for (String[] teacher : initialTeachers) {  originalTeacherData.add(teacher);  }   *// Инициализация модели таблицы учителей* teacherTableModel = new DefaultTableModel(teacherColumns, 0);  for (String[] teacher : originalTeacherData) {  teacherTableModel.addRow(teacher);  }  teacherTable = new JTable(teacherTableModel);  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"};  *// Исходные данные для таблицы учеников* String[][] initialStudents = {  {"Смирнов Алексей Иванович", "5А", "Отлично"},  {"Кузнецова Мария Петровна", "6Б", "Хорошо"},  {"Новиков Дмитрий Сергеевич", "7В", "Удовлетворительно"}  };  for (String[] student : initialStudents) {  originalStudentData.add(student);  }   *// Инициализация модели таблицы учеников* studentTableModel = new DefaultTableModel(studentColumns, 0);  for (String[] student : originalStudentData) {  studentTableModel.addRow(student);  }  studentTable = new JTable(studentTableModel);  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<>(new String[]{  "ФИО учителя", "Предмет", "Классы",  "ФИО ученика", "Класс ученика", "Успеваемость"  });  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();   *// Действие при нажатии кнопки "Поиск"* searchButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  String criterion = (String) searchCriteria.getSelectedItem();  String value = searchField.getText().trim();  searchTable(criterion, value);  }  });   *// Действие при нажатии кнопки "Сбросить"* resetButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  resetTable();  }  });   *// Действие при нажатии кнопки "Добавить учителя"* addTeacherButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  try {  *// Ввод и валидация ФИО учителя* String teacherName = promptForInput("Введите ФИО учителя:");  if (teacherName == null) return; *// Пользователь отменил ввод   // Ввод и валидация предмета* String subject = promptForInput("Введите предмет:");  if (subject == null) return;   *// Ввод и валидация классов* String classes = promptForInput("Введите классы (разделенные точкой с запятой ';'):");  if (classes == null) return;   *// Добавление нового учителя в таблицу и список исходных данных* String[] newTeacher = {teacherName, subject, classes};  teacherTableModel.addRow(newTeacher);  originalTeacherData.add(newTeacher);   } catch (InvalidInputException ex) {  *// Это исключение никогда не будет выброшено здесь, так как мы обрабатываем его в методе promptForInput* JOptionPane.*showMessageDialog*(frame, ex.getMessage(), "Ошибка ввода", JOptionPane.*ERROR\_MESSAGE*);  } catch (Exception ex) {  JOptionPane.*showMessageDialog*(frame, "Произошла непредвиденная ошибка: " + ex.getMessage(), "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  }  }  });   *// Действие при нажатии кнопки "Удалить учителя"* deleteTeacherButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  try {  int selectedRow = teacherTable.getSelectedRow();  if (selectedRow != -1) {  *// Преобразование индекса с учёта сортировки* selectedRow = teacherTable.convertRowIndexToModel(selectedRow);  teacherTableModel.removeRow(selectedRow);  originalTeacherData.remove(selectedRow);  } else {  throw new InvalidInputException("Пожалуйста, выберите учителя для удаления.");  }  } catch (InvalidInputException ex) {  JOptionPane.*showMessageDialog*(frame, ex.getMessage(), "Ошибка удаления", JOptionPane.*ERROR\_MESSAGE*);  } catch (Exception ex) {  JOptionPane.*showMessageDialog*(frame, "Произошла непредвиденная ошибка: " + ex.getMessage(), "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  }  }  });   *// Действие при нажатии кнопки "Добавить ученика"* addStudentButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  try {  *// Ввод и валидация ФИО ученика* String studentName = promptForInput("Введите ФИО ученика:");  if (studentName == null) return; *// Пользователь отменил ввод   // Ввод и валидация класса* String studentClass = promptForInput("Введите класс:");  if (studentClass == null) return;   *// Ввод и валидация успеваемости* String performance = promptForInput("Введите успеваемость:");  if (performance == null) return;   *// Добавление нового ученика в таблицу и список исходных данных* String[] newStudent = {studentName, studentClass, performance};  studentTableModel.addRow(newStudent);  originalStudentData.add(newStudent);   } catch (InvalidInputException ex) {  *// Это исключение никогда не будет выброшено здесь, так как мы обрабатываем его в методе promptForInput* JOptionPane.*showMessageDialog*(frame, ex.getMessage(), "Ошибка ввода", JOptionPane.*ERROR\_MESSAGE*);  } catch (Exception ex) {  JOptionPane.*showMessageDialog*(frame, "Произошла непредвиденная ошибка: " + ex.getMessage(), "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  }  }  });   *// Действие при нажатии кнопки "Удалить ученика"* deleteStudentButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  try {  int selectedRow = studentTable.getSelectedRow();  if (selectedRow != -1) {  *// Преобразование индекса с учёта сортировки* selectedRow = studentTable.convertRowIndexToModel(selectedRow);  studentTableModel.removeRow(selectedRow);  originalStudentData.remove(selectedRow);  } else {  throw new InvalidInputException("Пожалуйста, выберите ученика для удаления.");  }  } catch (InvalidInputException ex) {  JOptionPane.*showMessageDialog*(frame, ex.getMessage(), "Ошибка удаления", JOptionPane.*ERROR\_MESSAGE*);  } catch (Exception ex) {  JOptionPane.*showMessageDialog*(frame, "Произошла непредвиденная ошибка: " + ex.getMessage(), "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  }  }  });   *// Действие при нажатии кнопки "Загрузить данные"* loadButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  try {  loadDataFromFile();  } catch (DataLoadException ex) {  JOptionPane.*showMessageDialog*(frame, ex.getMessage(), "Ошибка загрузки", JOptionPane.*ERROR\_MESSAGE*);  } catch (Exception ex) {  JOptionPane.*showMessageDialog*(frame, "Произошла непредвиденная ошибка: " + ex.getMessage(), "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  }  }  });   *// Действие при нажатии кнопки "Сохранить данные"* saveButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  try {  saveDataToFile();  } catch (DataSaveException ex) {  JOptionPane.*showMessageDialog*(frame, ex.getMessage(), "Ошибка сохранения", JOptionPane.*ERROR\_MESSAGE*);  } catch (Exception ex) {  JOptionPane.*showMessageDialog*(frame, "Произошла непредвиденная ошибка: " + ex.getMessage(), "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  }  }  });   *// Делаем главное окно видимым* frame.setVisible(true);  }   */\*\*  \* 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*);  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));  }  } 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));  }  }  }   */\*\*  \* 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("");  }   */\*\*  \* Метод для загрузки данных из файла, выбранного пользователем.  \*  \* @throws DataLoadException Если возникает ошибка при загрузке данных.  \*/* private void loadDataFromFile() throws DataLoadException {  JFileChooser fileChooser = new JFileChooser();  fileChooser.setDialogTitle("Выберите текстовый файл для загрузки данных");  int userSelection = fileChooser.showOpenDialog(frame);   if (userSelection == JFileChooser.*APPROVE\_OPTION*) {  File fileToLoad = fileChooser.getSelectedFile();  try (BufferedReader br = new BufferedReader(new FileReader(fileToLoad))) {  String line;  boolean isTeacherSection = false;  boolean isStudentSection = false;   List<String[]> loadedTeachers = new ArrayList<>();  List<String[]> loadedStudents = new ArrayList<>();   while ((line = br.readLine()) != null) {  line = line.trim();  if (line.isEmpty()) continue;   if (line.equalsIgnoreCase("# Teachers")) {  isTeacherSection = true;  isStudentSection = false;  br.readLine(); *// Пропускаем заголовок столбцов* continue;  } else if (line.equalsIgnoreCase("# Students")) {  isTeacherSection = false;  isStudentSection = true;  br.readLine(); *// Пропускаем заголовок столбцов* continue;  }   if (isTeacherSection) {  String[] parts = line.split(",", 3);  if (parts.length == 3) {  loadedTeachers.add(new String[]{parts[0].trim(), parts[1].trim(), parts[2].trim()});  }  } else if (isStudentSection) {  String[] parts = line.split(",", 3);  if (parts.length == 3) {  loadedStudents.add(new String[]{parts[0].trim(), parts[1].trim(), parts[2].trim()});  }  }  }   *// Обновляем таблицы учителей* teacherTableModel.setRowCount(0);  originalTeacherData.clear();  for (String[] teacher : loadedTeachers) {  teacherTableModel.addRow(teacher);  originalTeacherData.add(teacher);  }   *// Обновляем таблицы учеников* studentTableModel.setRowCount(0);  originalStudentData.clear();  for (String[] student : loadedStudents) {  studentTableModel.addRow(student);  originalStudentData.add(student);  }   JOptionPane.*showMessageDialog*(frame, "Данные успешно загружены.", "Успех", JOptionPane.*INFORMATION\_MESSAGE*);   } catch (IOException ex) {  throw new DataLoadException("Ошибка при загрузке данных: " + ex.getMessage());  }  }  }   */\*\*  \* Метод для сохранения данных в файл, выбранный пользователем.  \*  \* @throws DataSaveException Если возникает ошибка при сохранении данных.  \*/* private void saveDataToFile() throws DataSaveException {  JFileChooser fileChooser = new JFileChooser();  fileChooser.setDialogTitle("Сохраните данные в текстовый файл");  int userSelection = fileChooser.showSaveDialog(frame);   if (userSelection == JFileChooser.*APPROVE\_OPTION*) {  File fileToSave = fileChooser.getSelectedFile();  try (BufferedWriter bw = new BufferedWriter(new FileWriter(fileToSave))) {  *// Записываем учителей* bw.write("# Teachers");  bw.newLine();  bw.write("ФИО учителя,Предмет,Классы");  bw.newLine();  for (String[] teacher : originalTeacherData) {  String line = String.*join*(",", teacher);  bw.write(line);  bw.newLine();  }   bw.newLine(); *// Пустая строка между разделами   // Записываем учеников* bw.write("# Students");  bw.newLine();  bw.write("ФИО ученика,Класс,Успеваемость");  bw.newLine();  for (String[] student : originalStudentData) {  String line = String.*join*(",", student);  bw.write(line);  bw.newLine();  }   JOptionPane.*showMessageDialog*(frame, "Данные успешно сохранены.", "Успех", JOptionPane.*INFORMATION\_MESSAGE*);   } catch (IOException ex) {  throw new DataSaveException("Ошибка при сохранении данных: " + ex.getMessage());  }  }  }   */\*\*  \* Метод для запроса ввода у пользователя с возможностью повторного ввода при ошибке.  \*  \* @param message Сообщение для отображения в диалоговом окне.  \* @return Введенное пользователем значение, либо null, если пользователь отменил ввод.  \* @throws InvalidInputException Если пользователь ввел пустую строку.  \*/* private String promptForInput(String message) throws InvalidInputException {  while (true) {  String input = JOptionPane.*showInputDialog*(frame, message);  if (input == null) {  *// Пользователь отменил ввод* return null;  }  input = input.trim();  if (input.isEmpty()) {  *// Показываем сообщение об ошибке и предлагаем повторный ввод* JOptionPane.*showMessageDialog*(frame, "Это поле не может быть пустым. Пожалуйста, введите значение.", "Ошибка ввода", JOptionPane.*ERROR\_MESSAGE*);  continue;  }  return input;  }  }   */\*\*  \* Точка входа в программу. Запуск приложения.  \*  \* @param args Аргументы командной строки (не используются).  \*/* public static void main(String[] args) {  *// Запуск интерфейса в потоке обработки событий Swing* SwingUtilities.*invokeLater*(new Runnable() {  public void run() {  new Main().SchoolManagementSystem();  }  });  } } |

# Приложение

Репозиторий: https://github.com/PlatonBarchenkov/OOP\_lab\_04.git