МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

КУРСОВАЯ РАБОТА

по дисциплине «Программирование на языке Java»

на тему **«Студенческая группа»**

|  |  |
| --- | --- |
| Выполнил: | студент группы 10702221  Будчанин В.А. |
| Руководитель: | доц. Сидорик В.В. |

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту (работе)

по дисциплине «Программирование на языке Java»

на тему **«Студенческая группа»**

|  |  |
| --- | --- |
| Выполнил: | студент группы 10702221  Будчанин В.А. |
| Руководитель: | доц. Сидорик В.В. |

Минск 2023

**Оглавление**

[Введение 2](#_Toc152527998)

[1. Постановка и описание задачи 3](#_Toc152527999)

[2. Теоретический раздел 4](#_Toc152528000)

[2.1 Язык программирования Java 4](#_Toc152528001)

[2.2 Библиотека Swing 5](#_Toc152528002)

[2.3 SQLite 6](#_Toc152528003)

[3. Математическая модель 7](#_Toc152528004)

[4. Реализация 8](#_Toc152528005)

[4.1 Построение UML диаграммы классов 8](#_Toc152528006)

[4.2 Реализация графических окон 11](#_Toc152528007)

[4.3 Стартовое окно с титульным листом 12](#_Toc152528008)

[4.4 Главные окна (с группами и студентами) 13](#_Toc152528009)

[4.5 Об авторе 16](#_Toc152528010)

[4.6 О программе 17](#_Toc152528011)

[4.7 Скриншоты с ошибками 18](#_Toc152528012)

[5. Результаты работы 19](#_Toc152528013)

[6. Описание структуры проекта 22](#_Toc152528014)

[Список использованной литературы 24](#_Toc152528015)

[Приложение 25](#_Toc152528016)

# Введение

У каждого старосты учебной группы так или иначе часто возникает необходимость в постоянном предоставлении информации о группе преподавателями или деканату, однако постоянно переделывать одни и те же данные под требования – дело, требующее время. Поэтому был реализован данный курсовой проект в целях минимизации временных затрат на создание и предоставление информации о студенческой группе.

Интерфейс программы позволяет производить ввод данных о студенческой группе и студентах, осуществлять просмотр имеющихся данных о студенческих группах и студентах, добавлять или удалять конкретные данные о студентах или группах, а также экспортировать данные в Excel и Word файлы в виде таблицы. Приложение имеет графический интерфейс пользователя, состоящий из меню, панели инструментов, таблицы и диалоговых окон. Приложение работает с базой данных SQLite.

В данной записке приведено описание программы «Студенческая группа», формирующей и использующей базу данных SQLite.

# Постановка и описание задачи

*Тема курсовой работы*: разработать приложение для управления студенческой группой с возможностью визуализации и сохранением данных. Программный код документировать. Сформировать с помощью утилиты javadoc html-файл, документирующий проект. Приложение скомпоновать в единый jar-файл, выполняемый на различных компьютерах.

*Цель программы*: получение данных о студентах в группах, визуальное отображение состава группы с возможностью сохранения данных о студентах в базе данных SQLite и экспорта этих данных в виде таблицы в Word и Excel.

*Исходные данные (информация о студентах и группах)*: фамилия, имя, отчество, номер телефона, места жительства (домашний адрес и текущий), проходящий обучение на платной основе или нет. Для групп необходим номер группы, курс и ФИО старосты.

*Выходные данные*: список студентов по группам.

В процессе работы над проектом были использованы следующие технологии:

1. Операционная система Windows 10, которая обеспечивает стабильность и совместимость с различными библиотеками и фреймворками;
2. Язык программирования Java, который позволяет создавать кроссплатформенные и масштабируемые приложения с высокой производительностью и безопасностью;
3. Платформа Swing, которая является частью стандартной библиотеки Java и предоставляет набор графических компонентов для разработки пользовательского интерфейса;
4. СУБД SQLite, которая представляет собой легковесную и встраиваемую базу данных, хранящую данные в виде файлов на диске;
5. Среда разработки Intellij IDEA 2023.2.3, которая обладает множеством функций для упрощения и ускорения процесса программирования, таких как подсветка синтаксиса, автодополнение, рефакторинг, отладка и тестирование кода.

# Теоретический раздел

## Язык программирования Java

Java — это многопоточный, компилируемый и объектно-ориентированный язык программирования, который широко применяется в различных сферах. Java поддерживает разработку приложений для разных платформ, включая настольные, веб- и мобильные устройства.

Java имеет большое распространение в мире. Это основной инструмент для создания Android-приложений. Он также используется в веб-разработке, государственных порталах и технологиях анализа больших данных. Java подходит для научных исследований, особенно в сфере обработки естественного языка. Язык Java доминировал в мобильном программировании еще до появления смартфонов — первые мобильные игры в начале 2000-х годов были созданы на Java.

Сферы применения Java:

* Веб-приложения. Java поддерживает множество технологий и фреймворков для создания динамичных и интерактивных веб-сайтов, таких как Servlets, JSP, JSF, Spring, Hibernate и т.д.
* Мобильные приложения. Java является основным языком для разработки приложений для платформы Android, которая доминирует на рынке смартфонов. Кроме того, Java может использоваться для создания кросс-платформенных мобильных приложений с помощью технологий, таких как PhoneGap, Cordova и Xamarin.
* Настольные приложения. Java предоставляет богатый набор графических компонентов и библиотек для создания пользовательских интерфейсов, таких как Swing, AWT, JavaFX и SWT. Java также позволяет интегрировать настольные приложения с другими системами и сервисами, используя технологии, такие как JDBC, RMI, JMS и SOAP.
* Игры и мультимедиа. Java обладает высокой производительностью и поддерживает разработку игр и мультимедийных приложений с использованием библиотек, таких как LWJGL, LibGDX, JOGL и JMonkeyEngine. Java также поддерживает работу с аудио и видео форматами, такими как MP3, WAV, OGG, MPEG и т.д.

## Библиотека Swing

Swing - это библиотека графического интерфейса пользователя (GUI) для Java, которая предоставляет набор компонентов для создания приложений с графическим интерфейсом. Swing является частью Java Foundation Classes (JFC), которые включают в себя также библиотеки AWT, Java 2D и Accessibility. Swing основан на AWT, но предлагает больше возможностей и гибкости для разработчиков GUI.

Основные преимущества Swing перед AWT заключаются в следующем:

- Swing компоненты не зависят от платформы, то есть они не используют родные ресурсы операционной системы для отрисовки своего внешнего вида. Это позволяет Swing компонентам иметь одинаковый вид и поведение на разных платформах и упрощает их кастомизацию.

- Swing компоненты поддерживают плоскую модель наследования, то есть все компоненты наследуются от одного базового класса JComponent. Это упрощает создание новых компонентов и расширение существующих.

- Swing компоненты поддерживают различные функции, такие как поддержка HTML и CSS, поддержка слушателей событий, поддержка внутренних фреймов, поддержка слоев, поддержка драг-энд-дроп и т.д.

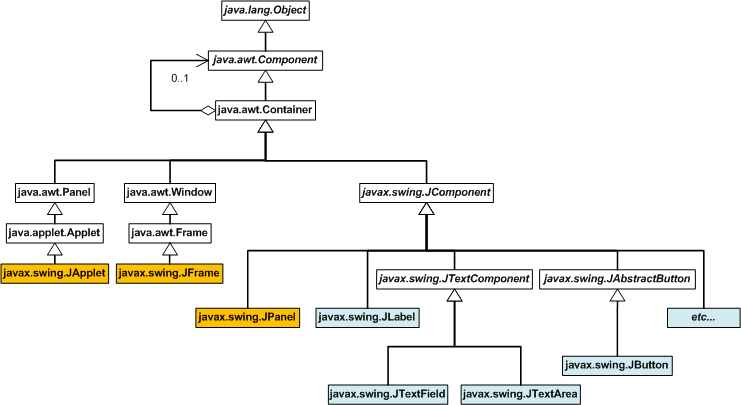


Рис. 1 – Иерархия наследования Swing от AWT

## SQLite

SQLite - это легковесная, встраиваемая и самодостаточная база данных SQL, которая не требует отдельного сервера для работы. Sqlite поддерживает большинство стандартных функций SQL, таких как транзакции, индексы, представления, триггеры и т.д. SQLite хранит все данные в одном файле, что облегчает перенос и резервное копирование данных. SQLite широко используется во многих приложениях, таких как браузеры, мессенджеры, электронные книги и мобильные устройства.

Особенности Sqlite включают в себя:

- Отсутствие необходимости в отдельном сервере баз данных, так как Sqlite хранит данные в обычных файлах на диске.

- Поддержка множества типов данных, таких как текст, числа, даты, булевы значения и BLOB.

- Поддержка транзакций с различными уровнями изоляции и возможностью отката.

- Низкое потребление ресурсов и высокая производительность при работе с небольшими и средними объемами данных.

SQLite - это удобный и надежный инструмент для разработки приложений, которые требуют локального хранения данных или синхронизации с удаленными серверами.

Чтобы использовать SQLite в java, необходимо установить специальный драйвер JDBC, который позволяет подключаться к базе данных и выполнять операции с данными.

Драйвер JDBC для Sqlite можно скачать с официального сайта или добавить в проект с помощью Maven или Gradle. После установки драйвера, можно создавать объекты Connection, Statement и ResultSet, как и при работе с другими базами данных.

# Математическая модель

Приложение содержит:

1. *Окно входа в приложение.* В котором титульный лист курсовой работы, реализовано начало работы с приложением и его завершение при необходимости либо при бездействии пользователя.
2. *Основное окно.* Окно для просмотра и взаимодействия с информацией, хранящейся в базе данных. В этом окне пользователь сможет внести данные о группе, используя текстовые поля.
3. *Окно со сведениями о студентах группы*. Окно для просмотра и взаимодействия с информацией о студентах, которая также хранится в базе данных. Студентов можно добавлять, удалять, редактировать данные о них. Предоставлена возможность экспортировать данные о них в Word и Excel для дальнейшего использования.
4. *Окно с информацией об авторе.* С актуальной информацией об авторе программы, контактные данные.
5. *Окно с информацией о программе.* С отображением информации о версии приложения, описание функционала программы, предоставляемой интерфейсом.

Для того, чтобы добавить группу, необходимо указать её номер, курс, ФИО старосты. Данные будут передаваться путем ввода информации в текстовые поля в основном окне программы для просмотра групп, предусмотрена валидация на пустой ввод в поля, а также проверка на наличие группы, в таком случае будет предлагаться удалить уже существующую группу.

Для того, чтобы добавить студента в соответствующем окне, необходимо как минимум указать обязательные сведения о нём: идентицирующий номер (следующий журналу или нет), ФИО, а также то, обучается ли на платной основе. В качестве дополнительной информации можно указать адрес домашний, текущий, номер телефона и информацию о том, местный студент или нет. Также предусмотрены проверки на существование студента в базе данных (по ключу идентификатор + группа) и ввод обязательных данных и т.д.

# Реализация

## Построение UML диаграммы классов

Унифицированный язык моделирования (UML) – это язык для специфицирования, визуализации, конструирования и документирования программных систем, а также бизнес-моделей и прочих не программных систем. При решении сложных задач заблаговременное планирование и моделирование значительно упрощают программирование.

Далее представлены UML-диаграммы проекта по частям.



Рисунок 3 – UML-диаграмма основных классов (группа и студент)

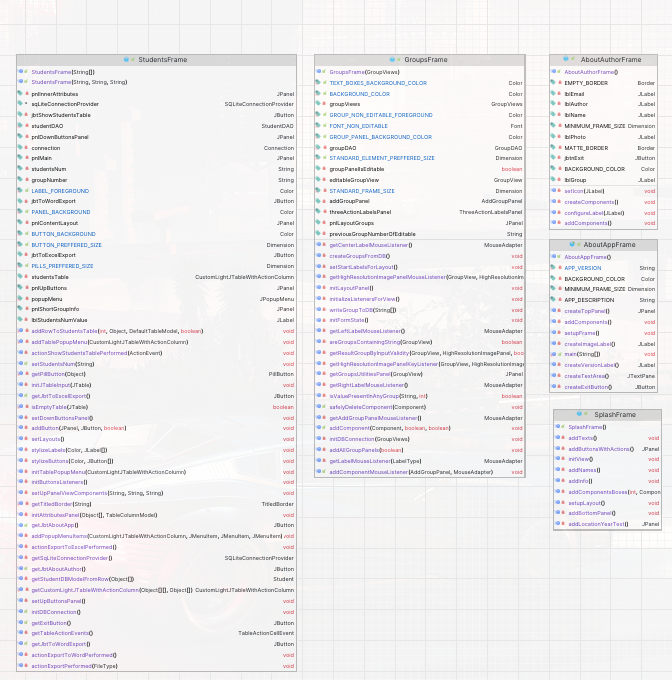


Рисунок 4 – UML-диаграмма классов для визуализирования хранимой информации

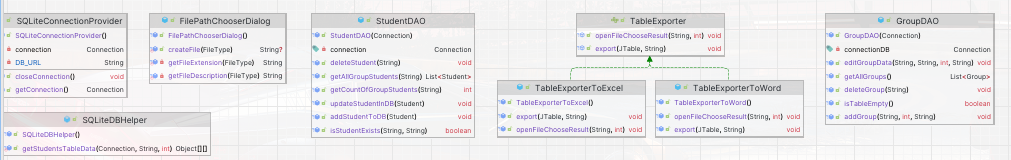


Рисунок 5 – UML-диаграмма классов для экспорта данных и сохранения в базу данных

## Реализация графических окон

Приложение состоит из 6 окон взаимосвязанных между собой:

* Splash Screen – титульный лист с возможностью выхода из приложения или переходом к окну с группами, если за 60 секунд это окно не былы закрыто, приложение завершает свою работу.
* Groups Window - одно из двух основных окон для работы с группами.
* Students Window – второе основное окно для работы с данными о студентах.
* About program (о программе)
* About Author (об авторе)

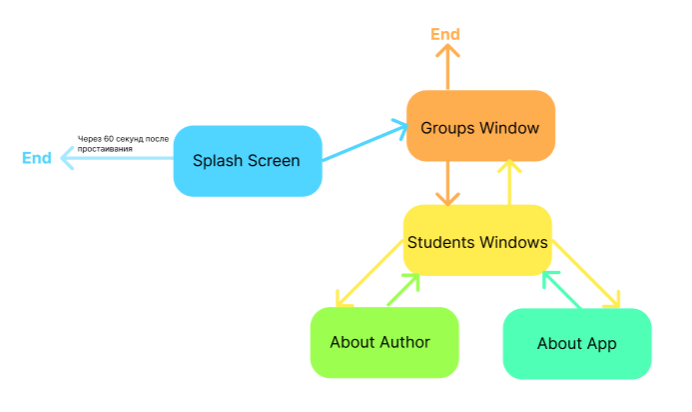


Рисунок 6 – Бизнес-логика переключения графических окон

## Стартовое окно с титульным листом

После запуска программы перед пользователем открывается стартовое окно. В стартовом окне отображена следующая информация: название учебного учреждения, кафедры, дисциплины, темы проекта, имени студента, выполнившего работу, и преподавателя, являющегося научным руководителем. Пользователь может выйти из программы, нажав кнопку «Выйти» или продолжить работу, нажав кнопку «Далее» (рис. 6.1). При бездействии пользователя в течение 60 секунд, программа закрывается.

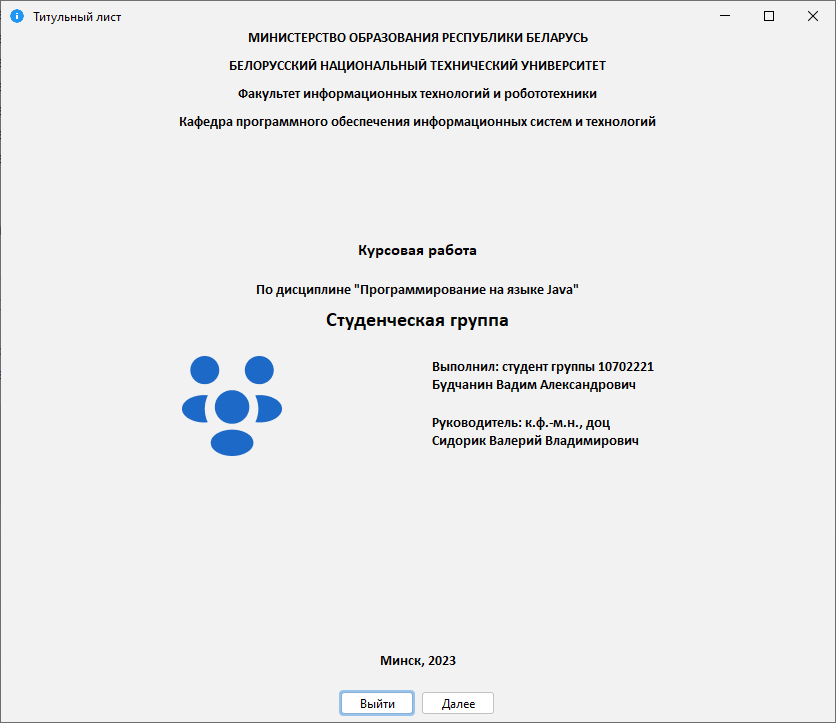


Рисунок 6.1 – Изображение стартового окна

## Главные окна (с группами и студентами)

При нажатии на кнопку «Далее» в основном окне перед пользователем открывается рабочее окно, содержащее информацию об имеющихся группах в базе данных и кнопка для добавления новых групп. На скриншоте ниже представлены две группы: 10702221 и 10702222. Группы можно раскрывать в отдельном окне для просмотра студентов (левая кнопка под группой), редактировать сведения о группе (центральная кнопка под группой), удалить сведения о группе (правая кнопка под группой). Также можно отсортировать (рис. 7.1) группы в соответствии со значениями конкретных полей (метки вверху).

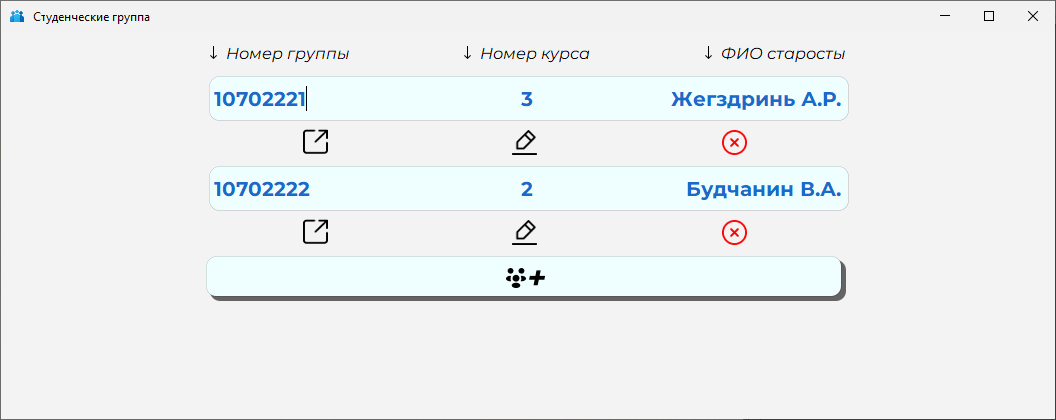


Рисунок 7 – Окно с группами

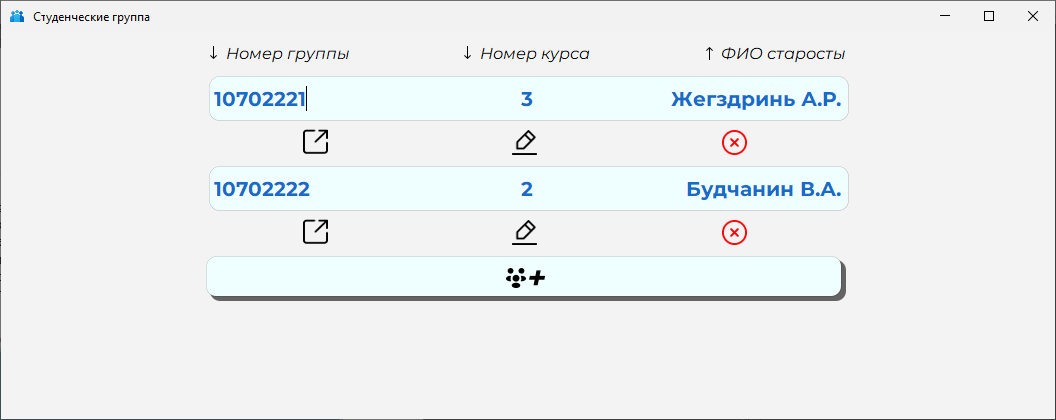


Рисунок 7.1 – Окно с отсортированными по ФИО старосты группами

При нажатии на левую кнопку под группой появляется окно с данными о студентах группы 10702221, где по группам выделены действия с показом и экспортом студентов, а также кнопки с дополнительной информацией и выходом из приложения (рис. 7.2).

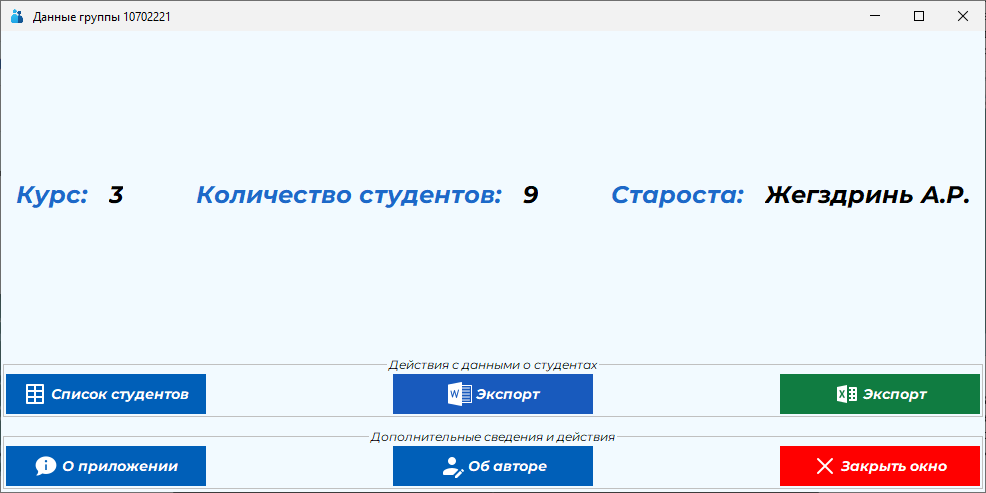


Рисунок 7.2 – Окно с данными о студентах группы

По нажатию на кнопку с надписью “Список студентов” пользователь видит таблицу со студентами и информацию о них. Здесь предоставлена возможность ввести данные о новом студенте и добавить их в базу данных, а также удалить студента. Для занесения результата действия в базу данных необходимо нажать правую кнопку в панели действия. Также имеются кнопки – фильтры, которые нужны для того, чтобы выводить конкретные столбцы и экспортировать или нет.

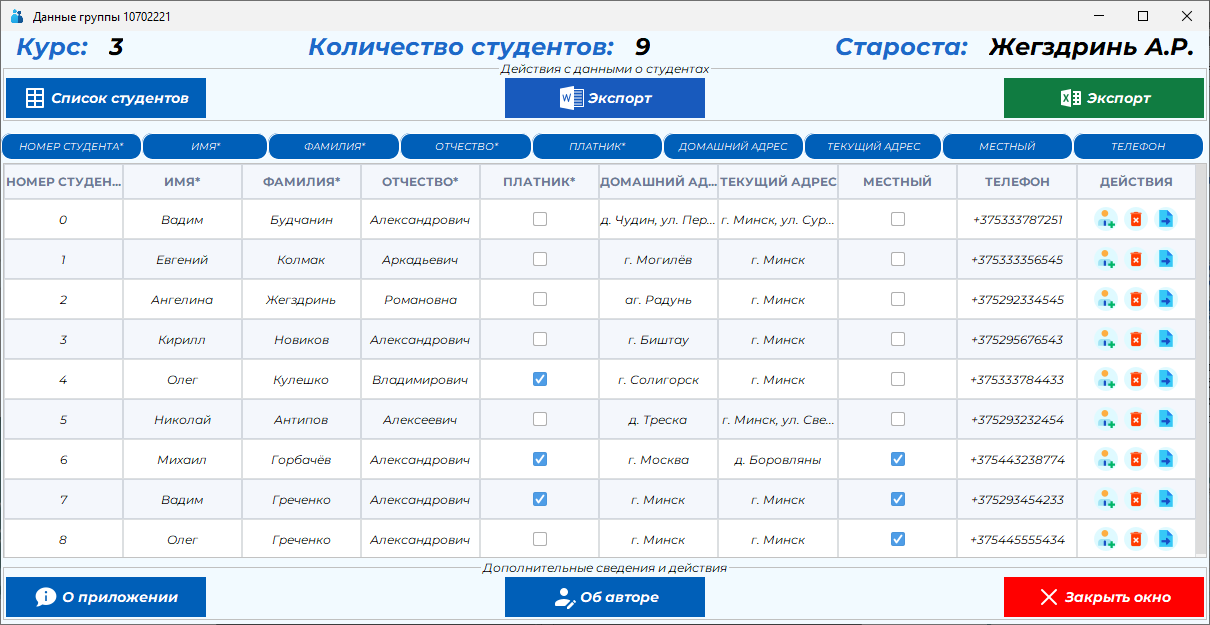


Рисунок 7.3 – Отображение информации о студентах

Ниже приведены скриншоты результата экспорта данных о студентах в Word (рис. 7.4) и в Excel (рис. 7.5)

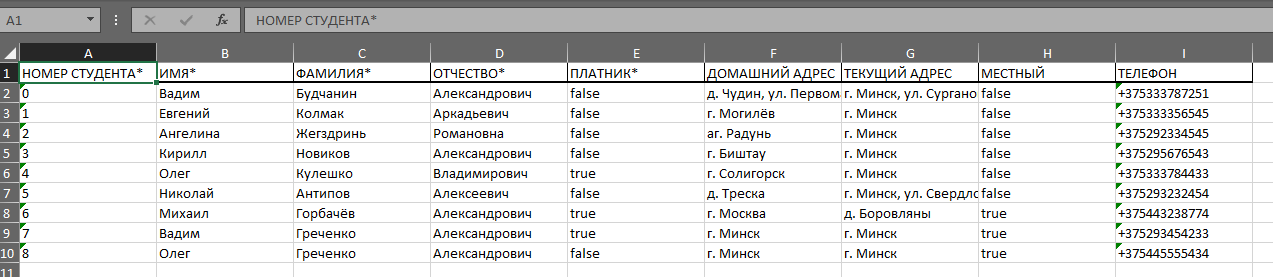


Рисунок 7.4 – Информация о студентах после экспорта в Excel

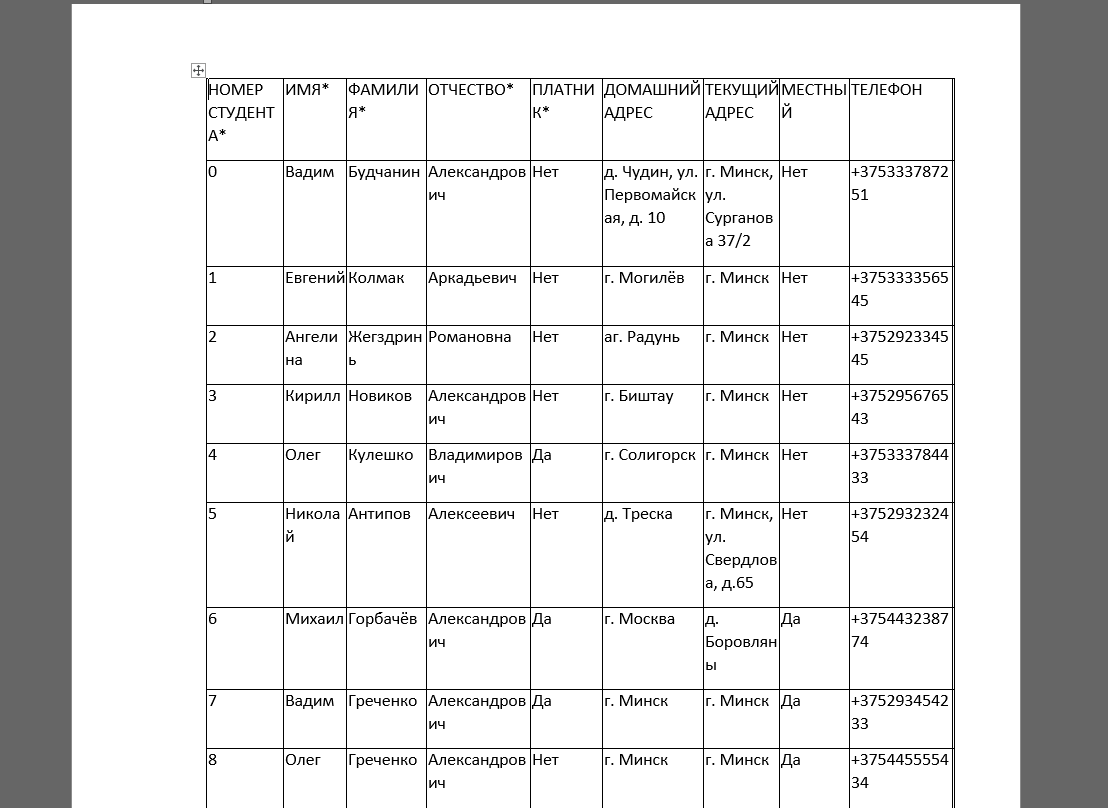


Рисунок 7.5 – Информация о студентах после экспорта в Word

## Об авторе

Окно «Об авторе» (рис. 8) содержит основную информацию об авторе проекта: номер группы, имя, фамилию, отчество, фотографию автора, а также его gmail.

Если данной информационное окно открыто, то пользователь не может взаимодействовать с другими окнами.



Рисунок 8 – Окно «Об авторе»

## О программе

Окно «О программе» (рис. 9) содержит основную информацию о возможностях программы и версию приложения.

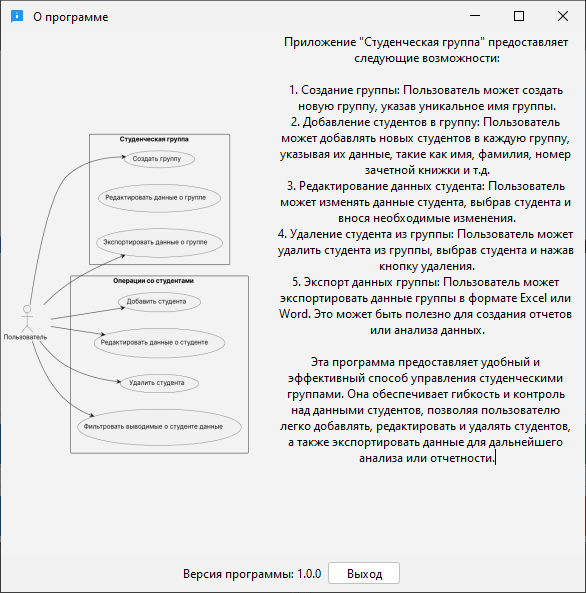


Рисунок 9 – Изображение окна «О программе»

## Скриншоты с ошибками

Текстовые поля выделяются красным, а текст становится белым после того, как пользователь попытался добавить группу хотя бы с одним не заполненным полем (рис. 10.1).

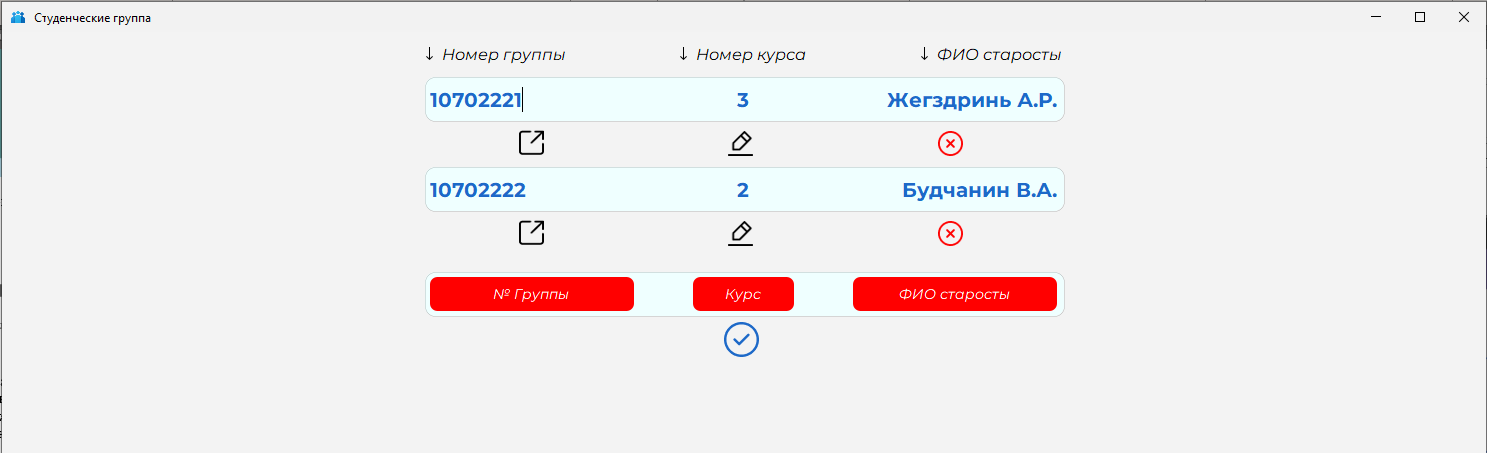


Рисунок 10.1 – Окно с сообщением об ошибке

При попытке добавления группы с существующим номером будет предложено удалить существующую группу или поменять номер группы (рис. 10.2).

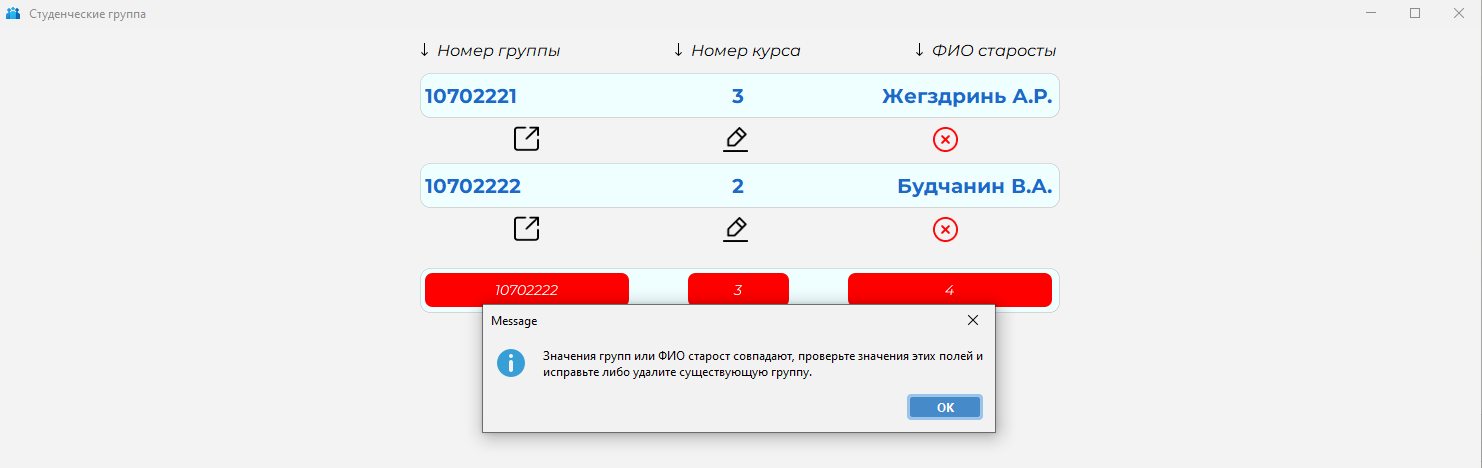


Рисунок 10.2 – Окно с сообщением об ошибке

Также предусмотрено совпадение номеров студентов, так, если номера совпадут, то будет предложена замена существующей записи или отмена добавления студента с последующей корректировкой значений (рис. 10.3).

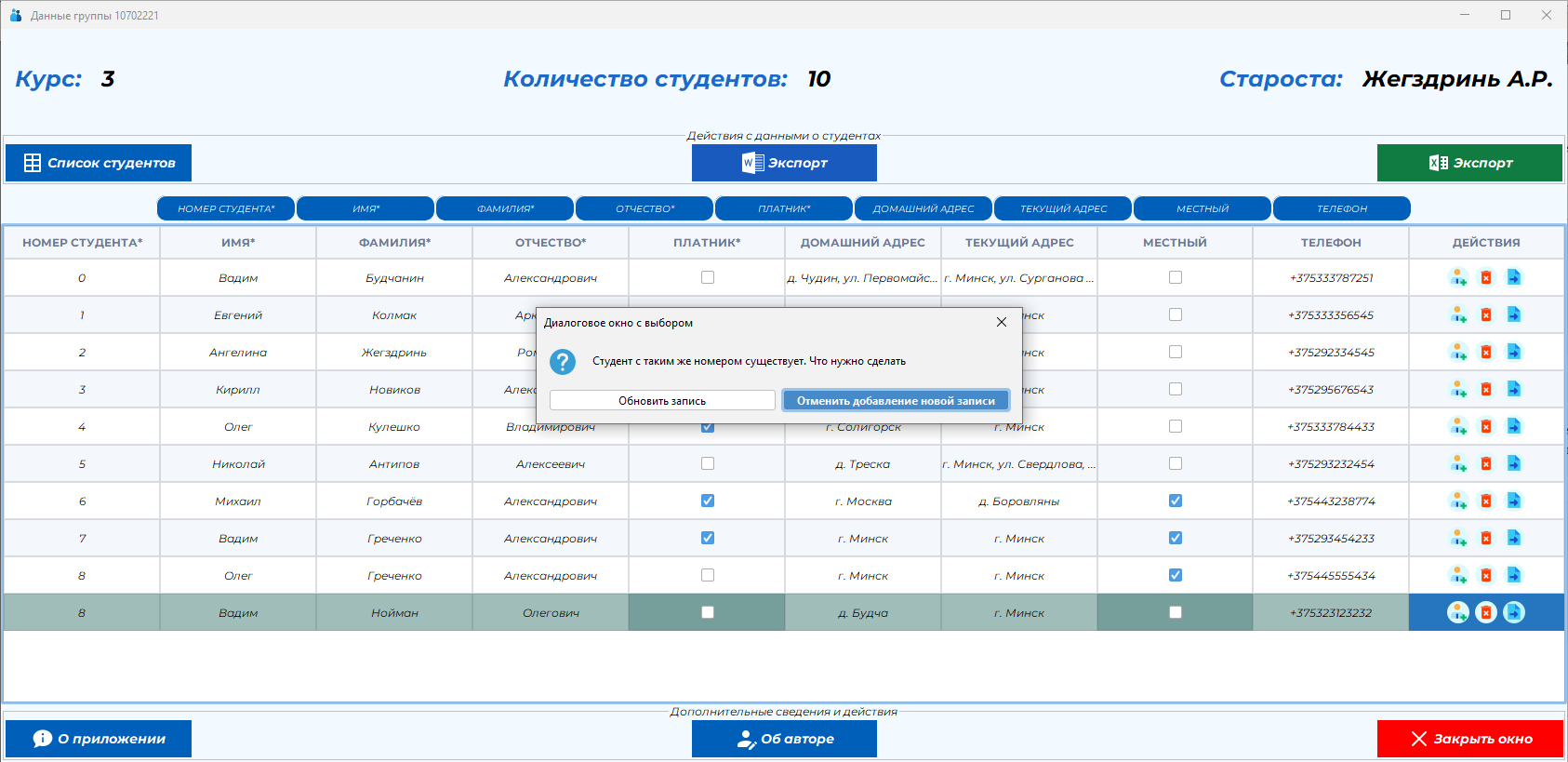


Рисунок 10.3 – Окно с сообщением об ошибке

# Результаты работы

Для начала работы с приложением необходимо добавить группу или несколько групп. Необходимо ввести данные в поля о группе (рис. 11.1):

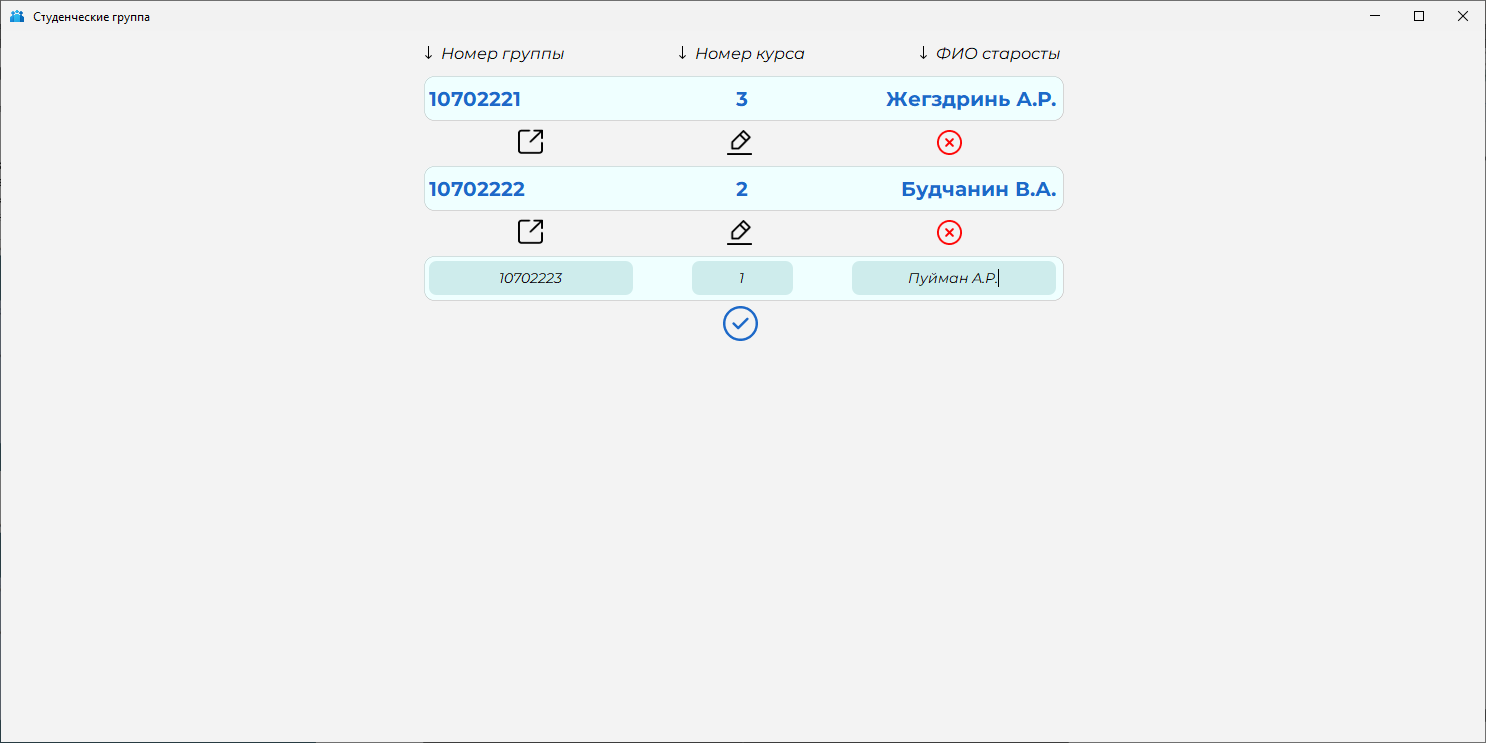


Рисунок 11.1 – Данные введены в текстовые поля

Далее необходимо подтвердить добавление, нажав галочку ниже, если совпадений не найдено и данные введены, то группа будет добавлена (рис. 11.2):

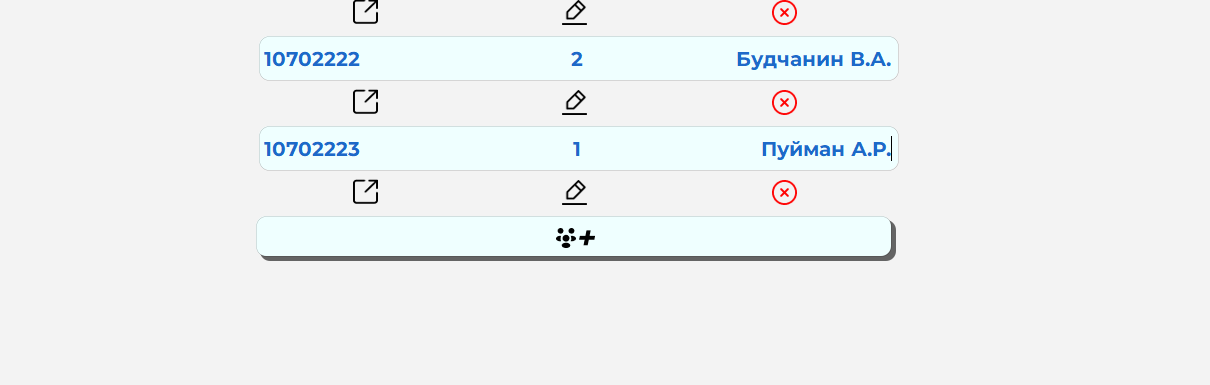


Рисунок 11.2 – Группа добавлена (10702223)

Открыв сведения о группе, увидим следующую картину (рис. 11.3):

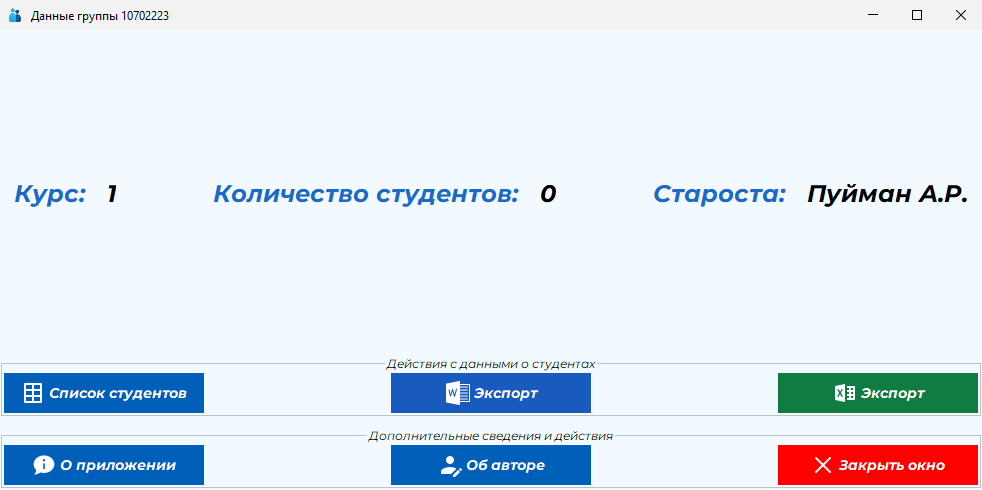


Рисунок 11.3 – Сведения о группе

Далее необходимо добавить студента или студентов, заполнив обязательные поля как минимум (рис. 11.4):

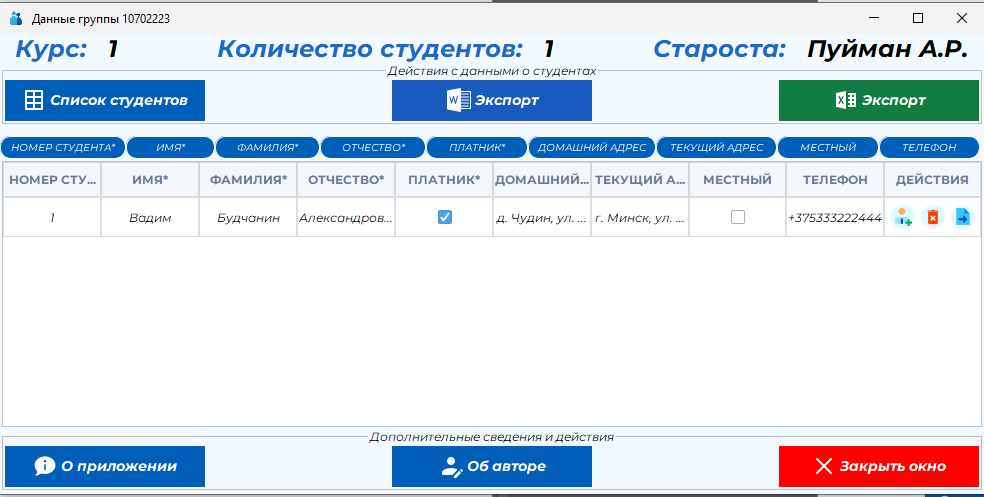


Рисунок 11.3 – Добавленный студент

Теперь таким образом, корректно вводя данные, можно добавлять группы, студентов в группы, а потом экспортировать данные в виде таблицы в Word или Excel файлы. При этом в приложении всегда будет храниться необходимая информация, сохранённая ранее.

# Описание структуры проекта

Структура проекта (рис. 5.1) была спроектирована в соответствии с требованиями к курсовому проекту.

В корневом каталоге app находятся:

1. zip – папка, в которой находится весь проект, упакованный в один архив zip;
2. doc – папка, в которой находится пояснительная записка в формате docx;
3. src – папка со всеми классами проекта;
4. javadoc – папка со сгенерированным JavaDoc документом.
5. dist – папка с jar-файлом проекта.
6. lib – папка с необходимыми библиотеками для реализации логики приложения (для работы с SQLite, Word, Excel).

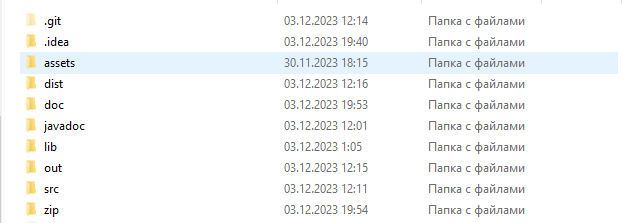


Рисунок 5.1 – необходимые директории.

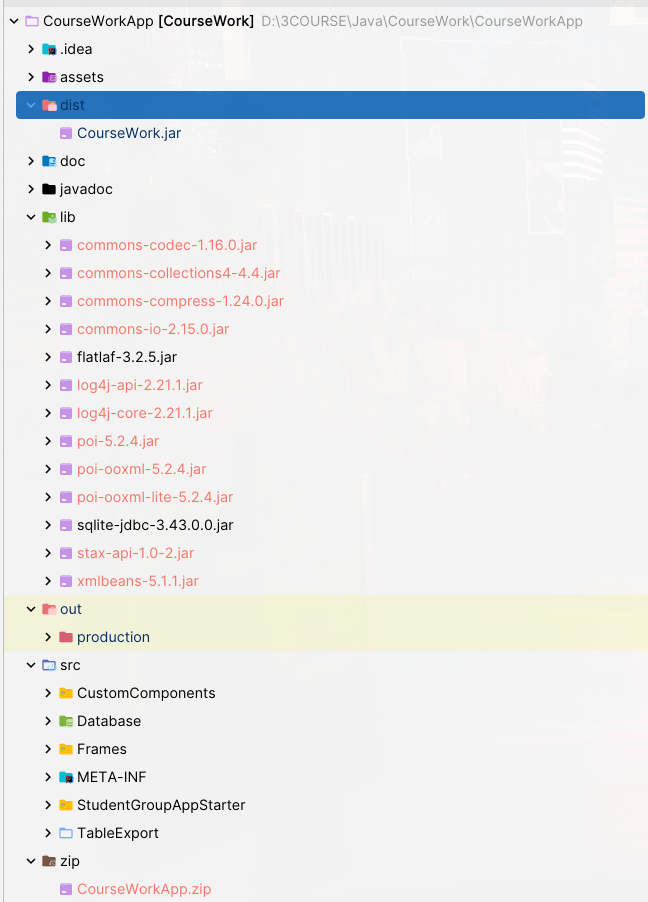


Рисунок 5.2 – Cтруктура проекта.

# Список использованной литературы

1. Horstmann, Cay S., 1959–Big Java : compatible with Java 5, 6 and 7 / Cay Horstmann. - 4th ed [Электронный ресурс] – Режим доступа: <https://pdfroom.com/books/big-java/qlgyyYDPgMG>
2. Блох, Джошуа Б70 Java: эффективное программирование, 3-е изд. : Пер. с англ. — СПб. : ООО “Диалектика”, 2019. [Электронный источник] – Режим доступа: <https://coderbooks.ru/books/java/java_ehffektivnoe_programmirovanie_3_izd_bloh_2019/>
3. Гуськова, Ольга Ивановна. Г968 Объектно ориентированное программирование в Java : учебное пособие / О. И. Гуськова. – Москва : МПГУ, 2018. [Электронный источник] – Режим доступа: <https://coderbooks.ru/books/java/obektno-orientirovannoe-programmirovanie-v-java/>
4. Шилдт, Герберт. Java. Полное руководство, 12-е изд. : Пер. с англ. - СПб. “Диалектика”; 2023. - 1344 с.: ил. - Парал. тит. Англ [Электронный ресурс] – Режим доступа: <https://coderbooks.ru/books/java/java_polnoe_rukovodstvo_12_izd_schildt_2023/>
5. The SQLite Handbook: A Start-to-Finish Resource for Learning and Using SQLite [Электронный ресурс]. – Режим доступа: <https://sqldocs.org/sqlite/introduction/>

# Приложение

Класс **SplashFrame**

package Frames;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
*/\*\*  
 \* Класс SplashFrame, представляющий титульный лист курсовой работы.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.0  
 \*/*public class SplashFrame extends JFrame {  
 */\*\*  
 \* Конструктор класса SplashFrame.  
 \*/* public SplashFrame() {  
 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 Dimension minimumSize = new Dimension(850, 730);  
 setMinimumSize(minimumSize);  
  
 Timer timer = new Timer(60000, e -> dispose());  
 timer.start();  
 setTitle("Титульный лист");  
 ImageIcon icon = new ImageIcon("assets/TitleFrameIcon.png");  
 setIconImage(icon.getImage());  
 setLocationRelativeTo(null);  
 initView();  
 setVisible(true);  
 }  
  
 */\*\*  
 \* Метод для инициализации интерфейса.  
 \*/* private void initView() {  
 setupLayout();  
 addNames();  
 addTexts();  
 addInfo();  
 addBottomPanel();  
 }  
  
 */\*\*  
 \* Метод для настройки макета.  
 \*/* private void setupLayout() {  
 setLayout(new BoxLayout(getContentPane(), BoxLayout.Y\_AXIS));  
 setBackground(new Color(243, 243, 243));  
 }  
  
 */\*\*  
 \* Метод для добавления названий.  
 \*/* private void addNames() {  
 String[] names = {"МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ",  
 "БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ",  
 "Факультет информационных технологий и робототехники",  
 "Кафедра программного обеспечения информационных систем и технологий"};  
 Font font = new Font("Calibri", Font.BOLD, 14);  
 for (String name : names) {  
 JLabel label = new JLabel(name);  
 label.setHorizontalAlignment(JLabel.CENTER);  
 label.setFont(font);  
 addComponentsBoxes(10, label);  
 }  
 }  
  
 */\*\*  
 \* Метод для добавления текстовых меток.  
 \*/* private void addTexts() {  
 add(Box.createVerticalStrut(100));  
  
 String[] texts = {"Курсовая работа",  
 "По дисциплине \"Программирование на языке Java\"",  
 "Студенческая группа"};  
  
 int[] sizes = {16, 14, 20};  
  
 for (int i = 0; i < texts.length; i++) {  
 JLabel label = new JLabel(texts[i]);  
 label.setFont(new Font("Calibri", Font.BOLD, sizes[i]));  
 addComponentsBoxes(i == 0 ? 20 : 10, label);  
 }  
 }  
  
 */\*\*  
 \* Метод для добавления информационных меток.  
 \*/* private void addInfo() {  
 String[] info = {"Выполнил: студент группы 10702221",  
 "Будчанин Вадим Александрович",  
 "Руководитель: к.ф.-м.н., доц",  
 "Сидорик Валерий Владимирович"};  
  
 JPanel pnlLabelsWithIcon = new JPanel(new FlowLayout(FlowLayout.CENTER, 150, 10));  
 ImageIcon icon = new ImageIcon("assets/BigGroupIcon.png");  
 JLabel lblIcon = new JLabel(icon);  
 JPanel jPnlLeft = new JPanel(new GridBagLayout());  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.gridwidth = GridBagConstraints.CENTER;  
 gbc.fill = GridBagConstraints.HORIZONTAL;  
 jPnlLeft.add(lblIcon, gbc);  
 pnlLabelsWithIcon.add(jPnlLeft);  
  
 JPanel jPanelRight = new JPanel();  
 jPanelRight.setLayout(new BoxLayout(jPanelRight, BoxLayout.Y\_AXIS));  
  
 for (int i = 0; i < info.length; i++) {  
 JLabel label = new JLabel(info[i]);  
 label.setFont(new Font("Calibri", Font.BOLD, 14));  
 jPanelRight.add(label);  
  
 if (i == 1) {  
 jPanelRight.add(Box.createVerticalStrut(20));  
 }  
 }  
  
 pnlLabelsWithIcon.add(jPanelRight);  
 add(pnlLabelsWithIcon);  
 }  
  
 */\*\*  
 \* Метод для добавления текста о местоположении и годе.  
 \*/* private JPanel addLocationYearText() {  
 JLabel jlblLocationYearText = new JLabel("Минск, 2023");  
 jlblLocationYearText.setFont(new Font("Calibri", Font.BOLD, 14));  
  
 JPanel bottomOutYearPanel = new JPanel(new BorderLayout());  
  
 JPanel pnlForLabel = new JPanel();  
 pnlForLabel.add(jlblLocationYearText);  
 bottomOutYearPanel.add(pnlForLabel, BorderLayout.NORTH);  
  
 return bottomOutYearPanel;  
 }  
  
 */\*\*  
 \* Метод для добавления кнопок с действиями.  
 \*/* private JPanel addButtonsWithActions() {  
 JButton jbtExit = new JButton("Выйти");  
 JButton jbtNext = new JButton("Далее");  
  
 jbtExit.addActionListener(e -> System.exit(0));  
  
 jbtNext.addActionListener(e -> {  
 GroupsFrame groupsFrame = new GroupsFrame(new GroupsFrame.GroupViews());  
 groupsFrame.setVisible(true);  
 dispose();  
 });  
  
 JPanel buttonsPanel = new JPanel();  
 JPanel bottomOutPanel = new JPanel(new BorderLayout());  
 buttonsPanel.setLayout(new FlowLayout());  
 buttonsPanel.add(jbtExit);  
 buttonsPanel.add(jbtNext);  
  
 bottomOutPanel.add(buttonsPanel, BorderLayout.SOUTH);  
 return bottomOutPanel;  
 }  
  
  
 */\*\*  
 \* Метод для добавления нижней панели.  
 \* Этот метод создает новую панель с BorderLayout.  
 \* Затем он добавляет результаты методов addLocationYearText() и addButtonsWithActions()  
 \* в северную и южную части этой панели соответственно.  
 \* Наконец, он добавляет эту панель в южную часть текущего компонента.  
 \*/* private void addBottomPanel() {  
 JPanel pnlGridBottom = new JPanel(new BorderLayout());  
 JPanel pnlGridContent = new JPanel(new GridLayout(2, 1));  
  
 pnlGridContent.add(addLocationYearText());  
 pnlGridContent.add(addButtonsWithActions());  
  
 pnlGridBottom.add(new JPanel(), BorderLayout.CENTER);  
 pnlGridBottom.add(pnlGridContent, BorderLayout.SOUTH);  
  
 add(pnlGridBottom, BorderLayout.SOUTH);  
 }  
  
 */\*\*  
 \* Метод для добавления компонентов в контейнеры Box.  
 \*  
 \* @param verticalStrut Размер вертикального пространства между компонентами.  
 \* @param components Компоненты для добавления.  
 \*/* private void addComponentsBoxes(int verticalStrut, Component... components) {  
 for (Component component : components) {  
 Box componentBox = Box.createHorizontalBox();  
 componentBox.add(Box.createHorizontalGlue());  
 componentBox.add(component);  
 componentBox.add(Box.createHorizontalGlue());  
 add(componentBox);  
 add(Box.createVerticalStrut(verticalStrut));  
 }  
 }  
}

Класс **GroupsFrame**

package Frames;  
  
import CustomComponents.\*;  
import Database.DAOS.GroupDAO;  
import Database.Managers.SQLiteConnectionProvider;  
import Frames.models.Group;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.sql.Connection;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Класс GroupsFrame представляет собой начальную форму создания учебной группы.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.4  
 \*/*public class GroupsFrame extends JFrame {  
 */\*\*  
 \* Предпочтительный размер стандартного элемента.  
 \*/* public static final Dimension *STANDARD\_ELEMENT\_PREFFERED\_SIZE* = new Dimension(640, 45);  
  
 */\*\*  
 \* Цвет фона.  
 \*/* public static final Color *BACKGROUND\_COLOR* = new Color(243, 243, 243);  
  
 */\*\*  
 \* Цвет фона панели группы.  
 \*/* public static final Color *GROUP\_PANEL\_BACKGROUND\_COLOR* = new Color(239, 255, 255);  
  
 */\*\*  
 \* Цвет текста неизменяемых полей группы.  
 \*/* public static final Color *GROUP\_NON\_EDITABLE\_FOREGROUND* = new Color(29, 105, 200);  
  
 */\*\*  
 \* Цвет фона текстовых полей.  
 \*/* public static final Color *TEXT\_BOXES\_BACKGROUND\_COLOR* = new Color(193, 228, 228, 179);  
  
 */\*\*  
 \* Шрифт для неизменяемых полей.  
 \*/* public static final Font *FONT\_NON\_EDITABLE* = new Font("Montserrat", Font.*BOLD*, 20);  
  
 */\*\*  
 \* Стандартный размер окна.  
 \*/* private static final Dimension *STANDARD\_FRAME\_SIZE* = new Dimension(1500, 750);  
  
 */\*\*  
 \* Панель для добавления группы.  
 \*/* private AddGroupPanel addGroupPanel;  
  
 */\*\*  
 \* DAO группы.  
 \*/* private static GroupDAO *groupDAO*;  
  
 */\*\*  
 \* Флаг, указывающий, является ли панель группы редактируемой.  
 \*/* private boolean groupPanelIsEditable = false;  
  
 */\*\*  
 \* Редактируемая панель ввода группы.  
 \*/* private GroupView editableGroupView = null;  
  
 */\*\*  
 \* Предыдущий номер группы для редактируемой панели.  
 \*/* private String previousGroupNumberOfEditable;  
  
 */\*\*  
 \* Модель групп.  
 \*/* private GroupViews groupViews;  
  
 */\*\*  
 \* Панель с тремя метками действий.  
 \*/* private final ThreeActionLabelsPanel threeActionLabelsPanel = new ThreeActionLabelsPanel(Color.*BLACK*,  
 GroupsFrame.*BACKGROUND\_COLOR*, "Номер группы", "Номер курса", "ФИО старосты");  
  
 */\*\*  
 \* Панель для размещения групп.  
 \*/* private JPanel pnlLayoutGroups;  
  
  
 */\*\*  
 \* Конструктор класса StartForm.  
 \*/* public GroupsFrame(GroupViews groupViews) {  
 initFormState();  
 initLayoutPanel();  
 initDBConnection(groupViews);  
 initializeListenersForView();  
 }  
  
 */\*\*  
 \* Инициализирует состояние формы.  
 \*/* private void initFormState() {  
 setTitle("Студенческие группа");  
 ImageIcon icon = new ImageIcon("assets/GroupIcon.png");  
 setIconImage(icon.getImage());  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setSize(*STANDARD\_FRAME\_SIZE*);  
 setPreferredSize(*STANDARD\_FRAME\_SIZE*);  
 setLocationRelativeTo(null);  
 pnlLayoutGroups = new JPanel();  
 }  
  
 */\*\*  
 \* Инициализирует панель макета.  
 \*/* private void initLayoutPanel() {  
 setLayout(new BorderLayout());  
 pnlLayoutGroups.setBackground(*BACKGROUND\_COLOR*);  
 pnlLayoutGroups.setLayout(new BoxLayout(pnlLayoutGroups, BoxLayout.*Y\_AXIS*));  
 JScrollPane scrollPane = new JScrollPane(pnlLayoutGroups);  
 scrollPane.createVerticalScrollBar();  
 scrollPane.setBorder(null);  
 add(scrollPane, BorderLayout.*CENTER*);  
 }  
  
 */\*\*  
 \* Добавляет компонент в макет.  
 \*  
 \* @param component компонент  
 \* @param hasVerticalStrutBefore флаг указывающий на необходимость добавления вертикального промежутка перед компонентом  
 \* @param hasVerticalStrutAfter флаг указывающий на необходимость добавления вертикального промежутка после компонента  
 \*/* public void addComponent(Component component, boolean hasVerticalStrutBefore, boolean hasVerticalStrutAfter) {  
 if (hasVerticalStrutBefore) {  
 pnlLayoutGroups.add(Box.*createVerticalStrut*(15));  
 }  
 pnlLayoutGroups.add(component);  
 if (hasVerticalStrutAfter) {  
 pnlLayoutGroups.add(Box.*createVerticalStrut*(15));  
 }  
 }  
  
 */\*\*  
 \* Добавляет слушатель событий мыши для панели добавления группы.  
 \*  
 \* @param addGroupPanel панель добавления группы  
 \* @param mouseAdapter слушатель событий мыши  
 \*/* public void addComponentMouseListener(AddGroupPanel addGroupPanel,  
 MouseAdapter mouseAdapter) {  
 addComponent(addGroupPanel, false, true);  
 addGroupPanel.setMouseClickEvent(mouseAdapter);  
 }  
  
 */\*\*  
 \* Безопасно удаляет компонент из макета.  
 \*  
 \* @param component компонент  
 \*/* public void safelyDeleteComponent(Component component) {  
 pnlLayoutGroups.remove(component);  
 pnlLayoutGroups.revalidate();  
 pnlLayoutGroups.repaint();  
 }  
  
 */\*\*  
 \* Метод для инициализации соединения с базой данных.  
 \*  
 \* @param groupViews Представления группы.  
 \*/* private void initDBConnection(GroupViews groupViews) {  
 SQLiteConnectionProvider sqLiteConnectionProvider = new SQLiteConnectionProvider();  
 Connection connection = null;  
 connection = sqLiteConnectionProvider.getConnection();  
 *groupDAO* = new GroupDAO(connection);  
  
 this.groupViews = groupViews;  
 Connection finalConnection = connection;  
 addWindowListener(new WindowAdapter() {  
 */\*\*  
 \* Вызывается в процессе закрытия окна formView.  
 \*  
 \* @param e - экземпляр WindowEvent  
 \*/* @Override  
 public void windowClosing(WindowEvent e) {  
 try {  
 if (finalConnection != null && !finalConnection.isClosed()) {  
 sqLiteConnectionProvider.closeConnection();  
 }  
 } catch (SQLException exception) {  
 exception.printStackTrace();  
 }  
 }  
 });  
 }  
  
 */\*\*  
 \* Инициализирует слушателей для представления.  
 \*/* private void initializeListenersForView() {  
 threeActionLabelsPanel.getLeftLabel().addMouseListener(getLeftLabelMouseListener());  
 threeActionLabelsPanel.getCenterLabel().addMouseListener(getCenterLabelMouseListener());  
 threeActionLabelsPanel.getRightLabel().addMouseListener(getRightLabelMouseListener());  
 threeActionLabelsPanel.setPreferredSize(GroupsFrame.*STANDARD\_ELEMENT\_PREFFERED\_SIZE*);  
 addComponent(threeActionLabelsPanel, false, false);  
 createGroupsFromDB();  
 addGroupPanel = new AddGroupPanel(Color.*BLACK*, GroupsFrame.*GROUP\_PANEL\_BACKGROUND\_COLOR*);  
 addGroupPanel.setToolTipText("Добавить новую группу");  
 addGroupPanel.setPreferredSize(GroupsFrame.*STANDARD\_ELEMENT\_PREFFERED\_SIZE*);  
 addComponentMouseListener(addGroupPanel, getAddGroupPanelMouseListener());  
 }  
  
 */\*\*  
 \* Возвращает слушатель событий мыши для меток.  
 \*  
 \* @param labelType тип метки  
 \* @return слушатель событий мыши  
 \*/* private MouseAdapter getLabelMouseListener(ThreeActionLabelsPanel.LabelType labelType) {  
 return new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if (groupViews.getGroupPanels().isEmpty()) {  
 return;  
 }  
 setStartLabelsForLayout();  
 switch (labelType) {  
 case *LEFT*:  
 GroupView.*sortGroupListByLeftText*(groupViews.getGroupPanels(), threeActionLabelsPanel.isLeftArrowDown());  
 threeActionLabelsPanel.setLeftLabelIcon(threeActionLabelsPanel.isLeftArrowDown() ?  
 ThreeActionLabelsPanel.*DOWN\_ARROW\_ICON* :  
 ThreeActionLabelsPanel.*UP\_ARROW\_ICON*);  
 break;  
 case *CENTER*:  
 GroupView.*sortPanelListByCenterText*(groupViews.getGroupPanels(), threeActionLabelsPanel.isCenterArrowDown());  
 threeActionLabelsPanel.setCenterLabelIcon(threeActionLabelsPanel.isCenterArrowDown() ?  
 ThreeActionLabelsPanel.*DOWN\_ARROW\_ICON* :  
 ThreeActionLabelsPanel.*UP\_ARROW\_ICON*);  
 break;  
 case *RIGHT*:  
 GroupView.*sortPanelListByRightText*(groupViews.getGroupPanels(), threeActionLabelsPanel.isRightArrowDown());  
 threeActionLabelsPanel.setRightLabelIcon(threeActionLabelsPanel.isRightArrowDown() ?  
 ThreeActionLabelsPanel.*DOWN\_ARROW\_ICON* :  
 ThreeActionLabelsPanel.*UP\_ARROW\_ICON*);  
 break;  
 }  
  
 addAllGroupPanels(false);  
 pnlLayoutGroups.revalidate();  
 pnlLayoutGroups.repaint();  
 }  
 };  
 }  
  
 */\*\*  
 \* Устанавливает начальные метки для макета.  
 \*/* private void setStartLabelsForLayout() {  
 pnlLayoutGroups.removeAll();  
 addComponent(threeActionLabelsPanel, false, false);  
 }  
  
 */\*\*  
 \* Добавляет все панели групп в макет.  
 \*  
 \* @param withClearing флаг указывающий на необходимость очистки макета перед добавлением  
 \*/* private void addAllGroupPanels(boolean withClearing) {  
 if (withClearing) {  
 setStartLabelsForLayout();  
 }  
 for (GroupView groupPanel : groupViews.getGroupPanels()) {  
 groupPanel.setPreferredSize(GroupsFrame.*STANDARD\_ELEMENT\_PREFFERED\_SIZE*);  
 addComponent(groupPanel, false, false);  
 JPanel utilitiesPanel = getGroupsUtilitiesPanel(groupPanel);  
 utilitiesPanel.setPreferredSize(GroupsFrame.*STANDARD\_ELEMENT\_PREFFERED\_SIZE*);  
 addComponent(utilitiesPanel, false, false);  
 addComponent(addGroupPanel, false, false);  
 }  
 pnlLayoutGroups.revalidate();  
 pnlLayoutGroups.repaint();  
 }  
  
 */\*\*  
 \* Возвращает слушателя событий мыши для левой метки.  
 \*  
 \* @return слушатель событий мыши  
 \*/* private MouseAdapter getLeftLabelMouseListener() {  
 return getLabelMouseListener(ThreeActionLabelsPanel.LabelType.*LEFT*);  
 }  
  
 */\*\*  
 \* Возвращает слушателя событий мыши для центральной метки.  
 \*  
 \* @return слушатель событий мыши  
 \*/* private MouseAdapter getCenterLabelMouseListener() {  
 return getLabelMouseListener(ThreeActionLabelsPanel.LabelType.*CENTER*);  
 }  
  
 */\*\*  
 \* Возвращает слушателя событий мыши для правой метки.  
 \*  
 \* @return слушатель событий мыши  
 \*/* private MouseAdapter getRightLabelMouseListener() {  
 return getLabelMouseListener(ThreeActionLabelsPanel.LabelType.*RIGHT*);  
 }  
  
 */\*\*  
 \* Возвращает слушателя событий мыши для панели добавления группы.  
 \*  
 \* @return слушатель событий мыши  
 \*/* private MouseAdapter getAddGroupPanelMouseListener() {  
 return new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 GroupView groupView = new GroupView(Color.*BLACK*, GroupsFrame.*GROUP\_PANEL\_BACKGROUND\_COLOR*, GroupsFrame.*TEXT\_BOXES\_BACKGROUND\_COLOR*, "№ Группы", "Курс", "ФИО старосты");  
 groupView.setPreferredSize(GroupsFrame.*STANDARD\_ELEMENT\_PREFFERED\_SIZE*);  
 addComponent(groupView, false, false);  
 addGroupPanel.repaint();  
 pnlLayoutGroups.revalidate();  
 HighResolutionImagePanel highResolutionImagePanel = new HighResolutionImagePanel(new HighResolutionImageLabel("Icons/Tick-Circle.png", 35, 35), 640, 45);  
 Action doComponentsReload = new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 addAllGroupPanels(true);  
 }  
 };  
 highResolutionImagePanel.getInputMap(JComponent.*WHEN\_IN\_FOCUSED\_WINDOW*).put(KeyStroke.*getKeyStroke*("ESCAPE"), "escape");  
 highResolutionImagePanel.getActionMap().put("escape", doComponentsReload);  
  
 highResolutionImagePanel.getInputMap().put(KeyStroke.*getKeyStroke*("ESC"), doComponentsReload);  
 addComponent(highResolutionImagePanel, false, false);  
 safelyDeleteComponent(addGroupPanel);  
 highResolutionImagePanel.setFocusable(true);  
 highResolutionImagePanel.addMouseListener(getHighResolutionImagePanelMouseListener(groupView, highResolutionImagePanel, false));  
 highResolutionImagePanel.addKeyListener(getHighResolutionImagePanelKeyListener(groupView, highResolutionImagePanel, false));  
 }  
 };  
 }  
  
 */\*\*  
 \* Возвращает слушателя событий мыши для панели с изображением высокого разрешения.  
 \*  
 \* @param groupView панель ввода группы  
 \* @param highResolutionImagePanel панель с изображением высокого разрешения  
 \* @param isEditMode флаг указывающий на режим редактирования  
 \* @return слушатель событий мыши  
 \*/* private MouseAdapter getHighResolutionImagePanelMouseListener(GroupView groupView, HighResolutionImagePanel highResolutionImagePanel, boolean isEditMode) {  
 return new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 getResultGroupByInputValidity(groupView, highResolutionImagePanel, isEditMode);  
 }  
 };  
 }  
  
 */\*\*  
 \* Возвращает слушателя событий клавиатуры для панели с изображением высокого разрешения.  
 \*  
 \* @param groupView панель ввода группы  
 \* @param highResolutionImagePanel панель с изображением высокого разрешения  
 \* @param isEditMode флаг указывающий на режим редактирования  
 \* @return слушатель событий клавиатуры  
 \*/* private KeyAdapter getHighResolutionImagePanelKeyListener(GroupView groupView, HighResolutionImagePanel highResolutionImagePanel, boolean isEditMode) {  
 return new KeyAdapter() {  
 @Override  
 public void keyPressed(KeyEvent e) {  
 if (e.getKeyCode() == KeyEvent.*VK\_ENTER*) {  
 getResultGroupByInputValidity(groupView, highResolutionImagePanel, isEditMode);  
 }  
 }  
 };  
 }  
  
 */\*\*  
 \* Устанавливает состояние обложки группы в необходимое на основе валидности ввода.  
 \*  
 \* @param groupView панель ввода группы  
 \* @param highResolutionImagePanel панель с изображением высокого разрешения  
 \* @param isEditMode флаг указывающий на режим редактирования  
 \*/* private void getResultGroupByInputValidity(GroupView groupView, HighResolutionImagePanel highResolutionImagePanel, boolean isEditMode) {  
 if (areGroupsContainingString(groupView) && !groupPanelIsEditable) {  
 groupView.setTextFieldsUnValid();  
 JOptionPane.*showMessageDialog*(this, "Значения групп или ФИО старост совпадают, проверьте значения этих полей и исправьте либо удалите существующую группу.");  
 return;  
 }  
 if (!groupView.isInputValid()) {  
 groupView.setTextFieldsUnValid();  
 return;  
 } else {  
 groupView.setTextFieldsValid();  
 }  
  
 groupView.setPanelNonEditableCustom(GroupsFrame.*GROUP\_PANEL\_BACKGROUND\_COLOR*, GroupsFrame.*GROUP\_NON\_EDITABLE\_FOREGROUND*, GroupsFrame.*FONT\_NON\_EDITABLE*);  
  
 if (!groupPanelIsEditable && !groupView.equals(editableGroupView)) {  
 writeGroupToDB(groupView.getTextFieldValues());  
 } else {  
 *groupDAO*.editGroupData(previousGroupNumberOfEditable, groupView.getTextFieldValues()[0],  
 Integer.*parseInt*(groupView.getTextFieldValues()[1]),  
 groupView.getTextFieldValues()[2]);  
 groupPanelIsEditable = false;  
 editableGroupView = null;  
 previousGroupNumberOfEditable = "";  
 }  
  
 if (!isEditMode) {  
 JPanel utilitiesPanel = getGroupsUtilitiesPanel(groupView);  
 utilitiesPanel.setPreferredSize(GroupsFrame.*STANDARD\_ELEMENT\_PREFFERED\_SIZE*);  
 safelyDeleteComponent(highResolutionImagePanel);  
 addComponent(utilitiesPanel, false, false);  
 addComponent(this.addGroupPanel, false, false);  
 groupViews.addGroupPanel(groupView);  
 } else {  
 safelyDeleteComponent(highResolutionImagePanel);  
 addAllGroupPanels(true);  
 }  
 }  
  
 */\*\*  
 \* Метод для записи информации о группе в базу данных.  
 \*  
 \* @param groupData Массив данных о группе, содержащий следующие элементы:  
 \* - groupNumber (номер группы)  
 \* - course (курс)  
 \* - headmanFullName (полное имя старосты)  
 \* @throws NumberFormatException Если при попытке преобразовать курс в целое число возникает ошибка  
 \*/* private void writeGroupToDB(String[] groupData) {  
 String groupNumber = groupData[0];  
 String course = groupData[1];  
 String headmanFullName = groupData[2];  
  
 try {  
 *groupDAO*.addGroup(groupNumber, Integer.*parseInt*(course), headmanFullName);  
 } catch (NumberFormatException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Возвращает панель утилит группы, также добавляет слушателей событый мыши для иконок,  
 \* с которыми можно взаимодействовать.  
 \*  
 \* @param groupView панель ввода группы  
 \* @return панель утилит группы  
 \*/* private JPanel getGroupsUtilitiesPanel(GroupView groupView) {  
 Dimension imageLabelSize = new Dimension(27, 27);  
 JPanel groupPanel = new JPanel();  
 groupPanel.setOpaque(false);  
 groupPanel.setLayout(new GridLayout(1, 3));  
  
 HighResolutionImageLabel jlblOpenGroupForm = new HighResolutionImageLabel("UtilitiesIcons/Open-Form-Icon.png", 25, 25);  
 jlblOpenGroupForm.setToolTipText("Открывает форму редактирования группы");  
 jlblOpenGroupForm.setPreferredSize(imageLabelSize);  
  
 HighResolutionImageLabel jlblDeleteGroup = new HighResolutionImageLabel("UtilitiesIcons/Delete-Icon.png", 25, 25);  
 jlblDeleteGroup.setToolTipText("Удалить все данные, связанные с данной группой");  
 jlblDeleteGroup.setPreferredSize(imageLabelSize);  
  
 HighResolutionImageLabel jlblEditGroupData = new HighResolutionImageLabel("UtilitiesIcons/Edit-Icon.png", 25, 25);  
 jlblEditGroupData.setToolTipText("Редактировать обложку данной группы");  
 jlblEditGroupData.setPreferredSize(imageLabelSize);  
  
 jlblDeleteGroup.addMouseListener(new MouseAdapter() {  
 */\*\*  
 \* Вызывается при щелчке мыши на компоненте jlblDeleteGroup.  
 \*  
 \* @param e - экземпляр MouseEvent  
 \*/* @Override  
 public void mouseClicked(MouseEvent e) {  
 int choice = JOptionPane.*showOptionDialog*(null, "Вы действительно хотите удалить все данные о группе?",  
 "Проверка", JOptionPane.*DEFAULT\_OPTION*, JOptionPane.*WARNING\_MESSAGE*, null, new Object[]{"Да", "Нет"}, "Да");  
 if (choice != 0) return;  
 *groupDAO*.deleteGroup(groupView.getTextFieldValues()[0]);  
 groupViews.getGroupPanels().remove(groupView);  
 addAllGroupPanels(true);  
 }  
 });  
  
 jlblOpenGroupForm.addMouseListener(new MouseAdapter() {  
 */\*\*  
 \* Вызывается при щелчке мыши на компоненте jlblOpenGroupForm.  
 \*  
 \* @param e - экземпляр MouseEvent  
 \*/* @Override  
 public void mouseClicked(MouseEvent e) {  
 StudentsFrame studentsFrame = new StudentsFrame(groupView.getTextFieldValues());  
 studentsFrame.setVisible(true);  
 }  
 });  
  
 jlblEditGroupData.addMouseListener(new MouseAdapter() {  
 */\*\*  
 \* Вызывается при щелчке мыши на компоненте jlblEditGroupData.  
 \*  
 \* @param e - экземпляр MouseEvent  
 \*/* @Override  
 public void mouseClicked(MouseEvent e) {  
 if (groupPanelIsEditable) {  
 JOptionPane.*showMessageDialog*(null, "Закончите редактирование предыдущей панели.");  
 return;  
 }  
 int editablePanelIndex = groupViews.getGroupPanels().indexOf(groupView);  
 setStartLabelsForLayout();  
 for (int i = 0; i < groupViews.getGroupPanels().size(); i++) {  
 if (i == editablePanelIndex) {  
 groupPanelIsEditable = true;  
 editableGroupView = groupViews.getGroupPanels().get(i);  
 previousGroupNumberOfEditable = editableGroupView.getTextFieldValues()[0];  
 addComponent(groupViews.getGroupPanels().get(i), false, true);  
 groupViews.getGroupPanels().get(i).setPanelEditableStandardValues();  
 HighResolutionImagePanel highResolutionImagePanel = new HighResolutionImagePanel(new HighResolutionImageLabel("Icons/Tick-Circle.png", 35, 35), 640, 45);  
 addComponent(highResolutionImagePanel, false, true);  
 safelyDeleteComponent(addGroupPanel);  
 highResolutionImagePanel.setFocusable(true);  
 highResolutionImagePanel.addMouseListener(getHighResolutionImagePanelMouseListener(groupViews.getGroupPanels().get(i), highResolutionImagePanel, true));  
 continue;  
 }  
 addComponent(groupViews.getGroupPanels().get(i), true, true);  
 JPanel utilitiesPanel = getGroupsUtilitiesPanel(groupViews.getGroupPanels().get(i));  
 addComponent(utilitiesPanel, false, false);  
 }  
 addComponent(addGroupPanel, false, true);  
 }  
 });  
  
 groupPanel.add(jlblOpenGroupForm);  
 groupPanel.add(jlblEditGroupData);  
 groupPanel.add(jlblDeleteGroup);  
 groupPanel.setMaximumSize(new Dimension(630, 45));  
  
 return groupPanel;  
 }  
  
 */\*\*  
 \* Проверяет, содержатся ли значения из groupView в других группах.  
 \*  
 \* @param groupView панель ввода группы  
 \* @return true, если значения не содержатся в других группах, иначе false  
 \*/* private boolean areGroupsContainingString(GroupView groupView) {  
 int numOfTextFields = groupView.getTextFieldValues().length;  
 final int courseNumberIndexSkip = 1;  
 String[] textFieldValues = groupView.getTextFieldValues();  
 for (int i = 0; i < numOfTextFields; i++) {  
 if (i == courseNumberIndexSkip) {  
 *//Значит, что лишь значения номера группы и старосты не должно повторяться у разных групп* continue;  
 }  
 if (isValuePresentInAnyGroup(textFieldValues[i], i)) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 */\*\*  
 \* Проверяет, содержится ли значение в любой из групп.  
 \*  
 \* @param value значение для проверки  
 \* @param index индекс значения в массиве  
 \* @return true, если значение содержится в любой из групп, иначе false  
 \*/* private boolean isValuePresentInAnyGroup(String value, int index) {  
 for (GroupView groupPanel : groupViews.getGroupPanels()) {  
 String[] groupTextFieldValues = groupPanel.getTextFieldValues();  
 if (value.equals(groupTextFieldValues[index])) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 */\*\*  
 \* Создает группы из базы данных.  
 \* Если таблица пустая, то ничего не делает.  
 \*/* private void createGroupsFromDB() {  
 if (*groupDAO*.isTableEmpty()) return;  
 List<Group> groupList;  
 groupList = *groupDAO*.getAllGroups();  
 for (Group group : groupList) {  
 GroupView groupView = new GroupView(Color.*BLACK*, GroupsFrame.*GROUP\_PANEL\_BACKGROUND\_COLOR*,  
 GroupsFrame.*TEXT\_BOXES\_BACKGROUND\_COLOR*, "№ группы", "Курс", "ФИО старосты");  
 groupView.setTextFieldsLeftToRightStrings(group.getGroupNumber(), String.*valueOf*(group.getCourseNumber()), group.getHeadmanFullName());  
 groupView.setTextFieldsValid();  
 groupView.setPanelNonEditableCustom(GroupsFrame.*GROUP\_PANEL\_BACKGROUND\_COLOR*, GroupsFrame.*GROUP\_NON\_EDITABLE\_FOREGROUND*, GroupsFrame.*FONT\_NON\_EDITABLE*);  
 groupView.setPreferredSize(GroupsFrame.*STANDARD\_ELEMENT\_PREFFERED\_SIZE*);  
 addComponent(groupView, false, false);  
 JPanel utilitiesPanel = getGroupsUtilitiesPanel(groupView);  
 utilitiesPanel.setPreferredSize(GroupsFrame.*STANDARD\_ELEMENT\_PREFFERED\_SIZE*);  
 addComponent(utilitiesPanel, false, false);  
 groupViews.addGroupPanel(groupView);  
 }  
 }  
  
 */\*\*  
 \* Класс GroupViews содержит список панелей групп.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.0  
 \*/* public static class GroupViews {  
 */\*\*  
 \* Список групп студентов-видов  
 \*/* private final List<GroupView> groupPanels;  
  
 */\*\*  
 \* Конструктор класса GroupViews.  
 \* Создает новый экземпляр GroupViews и инициализирует список groupPanels.  
 \*/* public GroupViews() {  
 groupPanels = new ArrayList<>();  
 }  
  
 */\*\*  
 \* Возвращает список панелей групп.  
 \*  
 \* @return список панелей групп  
 \*/* public List<GroupView> getGroupPanels() {  
 return groupPanels;  
 }  
  
 */\*\*  
 \* Добавляет панель группы в список.  
 \*  
 \* @param groupPanel панель группы для добавления  
 \*/* public void addGroupPanel(GroupView groupPanel) {  
 groupPanels.add(groupPanel);  
 }  
  
 }  
}

Класс **StudentsFrame**

package Frames;  
  
import CustomComponents.CustomLightJTableWithActionColumn;  
import CustomComponents.CustomTableActionCells.TableActionCellEvent;  
import CustomComponents.PillButton;  
import Database.DAOS.StudentDAO;  
import Database.Managers.SQLiteConnectionProvider;  
import Database.Managers.SQLiteDBHelper;  
import Frames.models.Student;  
import TableExport.FilePathChooserDialog;  
import TableExport.TableExporter;  
import TableExport.TableExporterToExcel;  
import TableExport.TableExporterToWord;  
  
import javax.swing.\*;  
import javax.swing.border.BevelBorder;  
import javax.swing.border.EmptyBorder;  
import javax.swing.border.TitledBorder;  
import javax.swing.plaf.FontUIResource;  
import javax.swing.table.DefaultTableModel;  
import javax.swing.table.TableColumn;  
import javax.swing.table.TableColumnModel;  
import javax.swing.table.TableRowSorter;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.sql.Connection;  
import java.sql.SQLException;  
import java.util.Objects;  
  
*/\*\*  
 \* Класс StudentsFrame представляет форму для отображения информации о студенческой группе.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.1 04.11.2023  
 \*/*public class StudentsFrame extends JFrame {  
 */\*\*  
 \* Предпочтительный размер кнопки.  
 \*/* public static final Dimension *BUTTON\_PREFFERED\_SIZE* = new Dimension(200, 40);  
  
 */\*\*  
 \* Предпочтительный размер элемента управления "Pills".  
 \*/* public static final Dimension *PILLS\_PREFFERED\_SIZE* = new Dimension(150, 30);  
  
 */\*\*  
 \* Цвет переднего плана метки.  
 \*/* public static final Color *LABEL\_FOREGROUND* = new Color(29, 105, 200);  
  
 */\*\*  
 \* Цвет фона панели.  
 \*/* public static final Color *PANEL\_BACKGROUND* = new Color(242, 250, 255);  
  
 */\*\*  
 \* Цвет фона кнопки.  
 \*/* public static final Color *BUTTON\_BACKGROUND* = new Color(0, 95, 184);  
 */\*\*  
 \* Всплывающее меню.  
 \*/* private static JPopupMenu *popupMenu*;  
 */\*\*  
 \* Объект доступа к данным студента.  
 \*/* private static StudentDAO *studentDAO*;  
 */\*\*  
 \* Главная панель.  
 \*/* private final JPanel pnlMain = new JPanel();  
 */\*\*  
 \* Внутреннее меню панели.  
 \*/* private final JPanel pnlUpButtons = new JPanel();  
 */\*\*  
 \* Внутренние атрибуты панели.  
 \*/* private final JPanel pnlInnerAttributes = new JPanel();  
 */\*\*  
 \* Внутренняя верхняя панель.  
 \*/* private final JPanel pnlShortGroupInfo = new JPanel();  
 */\*\*  
 \* Макет содержимого панели.  
 \*/* private final JPanel pnlContentLayout = new JPanel();  
 */\*\*  
 \* Нижняя панель, содержащая кнопки "Выход", "Об Авторе", "О Программе"  
 \*/* private final JPanel pnlDownButtonsPanel = new JPanel();  
 */\*\*  
 \* Номер группы.  
 \*/* private final String groupNumber;  
  
 */\*\*  
 \* Число студентов.  
 \*/* private final String studentsNum;  
 */\*\*  
 \* Провайдер соединения SQLite.  
 \*/* SQLiteConnectionProvider sqLiteConnectionProvider;  
 */\*\*  
 \* Значение числа студентов.  
 \*/* private JLabel lblStudentsNumValue;  
 */\*\*  
 \* Кнопка для отображения списка студентов.  
 \*/* private JButton jbtShowStudentsTable;  
 */\*\*  
 \* Кнопка для экспорта в Word.  
 \*/* private JButton jbtToWordExport;  
 */\*\*  
 \* Кнопка для экспорта в Excel.  
 \*/* private JButton jbtToExcelExport;  
 */\*\*  
 \* Таблица студентов с действием в столбце.  
 \*/* private CustomLightJTableWithActionColumn studentsTable = null;  
 */\*\*  
 \* Соединение с базой данных.  
 \*/* private Connection connection;  
  
 */\*\*  
 \* Создает новый экземпляр класса StudentsFrame с указанными номером группы, номером курса и ФИО старосты вместе с  
 \* кол-вом студентов в группе.  
 \*  
 \* @param groupNumber Номер группы.  
 \* @param courseNumber Номер курса.  
 \* @param headmanFullName ФИО старосты.  
 \*/* public StudentsFrame(String groupNumber, String courseNumber, String headmanFullName) {  
 *// Получаем провайдер соединения SQLite* sqLiteConnectionProvider = getSqLiteConnectionProvider();  
  
 *// Устанавливаем заголовок окна* setTitle("Данные группы " + groupNumber);  
  
 *// Устанавливаем номер группы и количество студентов* this.groupNumber = groupNumber;  
 this.studentsNum = String.*valueOf*(*studentDAO*.getCountOfGroupStudents(groupNumber));  
  
 *// Устанавливаем иконку окна* ImageIcon icon = new ImageIcon("assets/AloneGroupIcon.png");  
 setIconImage(icon.getImage());  
  
 *// Устанавливаем операцию закрытия окна, размер и расположение* setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
 setSize(1000, 500);  
 setLocationRelativeTo(null);  
  
 *// Настраиваем компоненты представления панели* setLayouts();  
 setUpPanelViewComponents(courseNumber, studentsNum, headmanFullName);  
  
 *// Настраиваем верхнюю панель кнопок* setUpButtonsPanel();  
  
 *// Настраиваем нижнюю панель кнопок* setDownButtonsPanel();  
  
 *// Инициализируем соединение с базой данных* initDBConnection();  
 }  
  
 */\*\*  
 \* Создает новый экземпляр класса StudentsFrame с указанными данными о группе (группа, курс и ФИО старосты) вместе с  
 \* кол-вом студентов в группе.  
 \*  
 \* @param groupData Массив данных о группе (группа, курс и ФИО старосты).  
 \*/* public StudentsFrame(String[] groupData) {  
 this(groupData[0], groupData[1], groupData[2]);  
 }  
  
 */\*\*  
 \* Возвращает кнопку-пилюлю.  
 \*  
 \* @param columnNames имена столбцов  
 \* @return кнопка-пилюля  
 \*/* private static PillButton getPillButton(Object columnNames) {  
 String columnName = columnNames.toString();  
 Color activatedBackground = new Color(0, 95, 184);  
 Color notActivatedBackground = new Color(243, 245, 246);  
 Color activatedForeground = Color.*WHITE*;  
 Color notActivatedForeground = Color.*BLACK*;  
 return new PillButton(columnName, activatedBackground, notActivatedBackground, activatedForeground, notActivatedForeground);  
 }  
  
 */\*\*  
 \* Проверяет, является ли таблица пустой.  
 \*  
 \* @param studentsTable таблица студентов  
 \* @return true, если таблица пустая, иначе false  
 \*/* private static boolean isEmptyTable(JTable studentsTable) {  
 return ((DefaultTableModel) studentsTable.getModel()).getRowCount() == 0;  
 }  
  
 */\*\*  
 \* Добавляет всплывающее контекстное меню к таблице.  
 \*  
 \* @param studentsTable таблица студентов  
 \*/* private static void addTablePopupMenu(CustomLightJTableWithActionColumn studentsTable) {  
 studentsTable.addMouseListener(new MouseAdapter() {  
 */\*\*  
 \* Вызывается при нажатии кнопки мыши.  
 \*  
 \* @param e экземпляр MouseEvent  
 \*/* public void mousePressed(MouseEvent e) {  
 showPopupMenu(e);  
 }  
  
 */\*\*  
 \* Вызывается при отпускании кнопки мыши.  
 \*  
 \* @param e экземпляр MouseEvent  
 \*/* public void mouseReleased(MouseEvent e) {  
 showPopupMenu(e);  
 }  
  
 */\*\*  
 \* Отображает всплывающее контекстное меню.  
 \*  
 \* @param e экземпляр MouseEvent  
 \*/* private void showPopupMenu(MouseEvent e) {  
 if (e.isPopupTrigger()) {  
 *popupMenu*.show(e.getComponent(), e.getX(), e.getY());  
 }  
 }  
 });  
 }  
  
 */\*\*  
 \* Возвращает настраиваемую таблицу студентов с действием в столбце.  
 \*  
 \* @param studentsData данные студентов  
 \* @param columnsNames названия столбцов  
 \* @return настраиваемая таблица студентов  
 \*/* private static CustomLightJTableWithActionColumn getCustomLightJTableWithActionColumn(Object[][] studentsData, Object[] columnsNames) {  
 DefaultTableModel defaultTableModel = new DefaultTableModel(studentsData, columnsNames);  
 CustomLightJTableWithActionColumn studentsTable = new CustomLightJTableWithActionColumn(defaultTableModel) {  
 @Override  
 public Class<?> getColumnClass(int column) {  
 return switch (column) {  
 case 0 -> Integer.class;  
 case 1, 2, 3, 5, 6, 8 -> String.class;  
 default -> Boolean.class;  
 };  
 }  
 };  
 TableRowSorter<DefaultTableModel> rowSorter = new TableRowSorter<>(defaultTableModel);  
 studentsTable.setRowSorter(rowSorter);  
 return studentsTable;  
 }  
  
 */\*\*  
 \* Добавляет строку в таблицу студентов.  
 \*  
 \* @param rowIndex индекс строки  
 \* @param newRowStudentID идентификатор новой строки студента  
 \* @param tableModel модель таблицы  
 \* @param isFirstRow флаг, указывающий, является ли это первой строкой  
 \*/* private static void addRowToStudentsTable(int rowIndex, Object newRowStudentID, DefaultTableModel tableModel, boolean isFirstRow) {  
 Object[] rowData = {newRowStudentID, "", "", "", false, "", "", false, ""};  
 int newStudentIndex = isFirstRow ? 0 : rowIndex + 1;  
 tableModel.insertRow(newStudentIndex, rowData);  
 tableModel.fireTableRowsInserted(newStudentIndex, newStudentIndex);  
 }  
  
 */\*\*  
 \* Метод setLayouts устанавливает компоновки для панелей.  
 \*/* private void setLayouts() {  
 pnlMain.setLayout(new BoxLayout(pnlMain, BoxLayout.*Y\_AXIS*));  
 pnlUpButtons.setLayout(new BoxLayout(pnlUpButtons, BoxLayout.*X\_AXIS*));  
 pnlInnerAttributes.setLayout(new BoxLayout(pnlInnerAttributes, BoxLayout.*X\_AXIS*));  
 pnlShortGroupInfo.setLayout(new BoxLayout(pnlShortGroupInfo, BoxLayout.*X\_AXIS*));  
 pnlContentLayout.setLayout(new BoxLayout(pnlContentLayout, BoxLayout.*Y\_AXIS*));  
 pnlDownButtonsPanel.setLayout(new BoxLayout(pnlDownButtonsPanel, BoxLayout.*X\_AXIS*));  
  
 pnlMain.add(pnlShortGroupInfo);  
 pnlMain.add(pnlUpButtons);  
 pnlMain.add(Box.*createVerticalStrut*(10));  
 pnlMain.add(pnlInnerAttributes);  
 pnlMain.add(pnlContentLayout);  
 pnlMain.add(pnlDownButtonsPanel);  
 pnlMain.setBackground(*PANEL\_BACKGROUND*);  
 pnlInnerAttributes.setBackground(*PANEL\_BACKGROUND*);  
 pnlContentLayout.setBackground(*PANEL\_BACKGROUND*);  
  
 add(pnlMain);  
 setContentPane(pnlMain);  
 }  
  
 */\*\*  
 \* Метод setUpPanelViewComponents устанавливает компоненты представления панели.  
 \*  
 \* @param course Курс.  
 \* @param studentsCount Количество студентов.  
 \* @param headmanFullName ФИО старосты.  
 \*/* private void setUpPanelViewComponents(String course, String studentsCount, String headmanFullName) {  
 int strutWidth = 15;  
  
 JLabel lblCourse = new JLabel("Курс: ");  
 JLabel lblStudentsNum = new JLabel("Количество студентов: ");  
 JLabel lblHeadmanFullName = new JLabel("Староста: ");  
 JLabel lblCourseValue = new JLabel(course);  
 JLabel lblHeadmanValue = new JLabel(headmanFullName);  
 lblStudentsNumValue = new JLabel(studentsCount);  
  
 pnlShortGroupInfo.setLayout(new BoxLayout(pnlShortGroupInfo, BoxLayout.*X\_AXIS*));  
  
 stylizeLabels(*LABEL\_FOREGROUND*, lblCourse, lblStudentsNum, lblHeadmanFullName);  
 stylizeLabels(Color.*BLACK*, lblCourseValue, lblStudentsNumValue, lblHeadmanValue);  
  
 pnlShortGroupInfo.add(Box.*createHorizontalStrut*(strutWidth));  
 pnlShortGroupInfo.add(lblCourse);  
  
 pnlShortGroupInfo.add(Box.*createHorizontalStrut*(strutWidth));  
 pnlShortGroupInfo.add(lblCourseValue);  
  
 pnlShortGroupInfo.add(Box.*createHorizontalStrut*(strutWidth));  
 pnlShortGroupInfo.add(Box.*createHorizontalGlue*());  
 pnlShortGroupInfo.add(lblStudentsNum);  
  
 pnlShortGroupInfo.add(Box.*createHorizontalStrut*(strutWidth));  
 pnlShortGroupInfo.add(lblStudentsNumValue);  
  
 pnlShortGroupInfo.add(Box.*createHorizontalStrut*(strutWidth));  
 pnlShortGroupInfo.add(Box.*createHorizontalGlue*());  
 pnlShortGroupInfo.add(lblHeadmanFullName);  
  
 pnlShortGroupInfo.add(Box.*createHorizontalStrut*(strutWidth));  
 pnlShortGroupInfo.add(lblHeadmanValue);  
 pnlShortGroupInfo.add(Box.*createHorizontalStrut*(strutWidth));  
  
 pnlShortGroupInfo.setBackground(*PANEL\_BACKGROUND*);  
 }  
 */\*\*  
 \* Метод getJbtAboutAuthor создает кнопку "Об авторе".  
 \*  
 \* @return JButton, который отображает информацию об авторе при нажатии.  
 \*/* public JButton getJbtAboutAuthor() {  
 JButton jbtAboutAuthorInfo = new JButton("Об авторе", new ImageIcon(Objects.*requireNonNull*(getClass()  
 .getResource("/CustomComponents/Icons/AuthorButtonIcon.png"))));  
 jbtToWordExport.setToolTipText("Показать краткие сведения об авторе приложения");  
 jbtAboutAuthorInfo.addActionListener(e -> {  
 AboutAuthorFrame aboutAuthorFrame = new AboutAuthorFrame();  
 aboutAuthorFrame.setLocationRelativeTo(null);  
 });  
 return jbtAboutAuthorInfo;  
 }  
  
 */\*\*  
 \* Метод getJbtAboutApp создает кнопку "О приложении".  
 \*  
 \* @return JButton, который отображает информацию о приложении при нажатии.  
 \*/* public JButton getJbtAboutApp() {  
 JButton jbtAboutAppInfo = new JButton("О приложении", new ImageIcon(Objects.*requireNonNull*(getClass()  
 .getResource("/CustomComponents/Icons/AboutButtonIcon.png"))));  
 jbtAboutAppInfo.setToolTipText("Показать краткую информацию о приложении");  
 jbtAboutAppInfo.addActionListener(e -> {  
 AboutAppFrame aboutAppFrame = new AboutAppFrame();  
 aboutAppFrame.setLocationRelativeTo(null);  
 });  
 return jbtAboutAppInfo;  
 }  
  
 */\*\*  
 \* Метод getExitButton создает кнопку "Закрыть окно".  
 \*  
 \* @return JButton, который закрывает окно при нажатии.  
 \*/* public JButton getExitButton() {  
 JButton jbtExit = new JButton("Закрыть окно", new ImageIcon(Objects.*requireNonNull*(getClass()  
 .getResource("/CustomComponents/Icons/CloseButtonIcon.png"))));  
 jbtExit.setToolTipText("Закрывает окно редактирования данной группы");  
 jbtExit.addActionListener(e -> {  
 JFrame jFrame = (JFrame) jbtExit.getTopLevelAncestor();  
 jFrame.dispose();  
 });  
 return jbtExit;  
 }  
  
 */\*\*  
 \* Метод setUpButtonsPanel устанавливает верхнюю панель кнопок.  
 \*/* private void setUpButtonsPanel() {  
 jbtShowStudentsTable = new JButton("Список студентов", new ImageIcon(Objects.*requireNonNull*(getClass()  
 .getResource("/CustomComponents/Icons/TableButtonIcon.png"))));  
 jbtShowStudentsTable.setToolTipText("Вывести список всех студентов для данной группы");  
  
 jbtToWordExport = new JButton("Экспорт", new ImageIcon(Objects.*requireNonNull*(getClass()  
 .getResource("/CustomComponents/Icons/WordIcon.png"))));  
  
 jbtToExcelExport = new JButton("Экспорт", new ImageIcon(Objects.*requireNonNull*(getClass()  
 .getResource("/CustomComponents/Icons/ExcelIcon.png"))));  
 jbtToExcelExport.setToolTipText("Экспортирует таблицу со студентами в Excel");  
  
 stylizeButtons(*BUTTON\_BACKGROUND*, jbtShowStudentsTable);  
 stylizeButtons(new Color(24, 90, 189), jbtToWordExport);  
 stylizeButtons(new Color(16, 124, 65), jbtToExcelExport);  
  
 pnlUpButtons.setBackground(*PANEL\_BACKGROUND*);  
 pnlUpButtons.setBorder(*getTitledBorder*("Действия с данными о студентах"));  
 pnlInnerAttributes.setBackground(*PANEL\_BACKGROUND*);  
 pnlContentLayout.setBackground(*PANEL\_BACKGROUND*);  
 addButton(pnlUpButtons, jbtShowStudentsTable, false);  
 addButton(pnlUpButtons, jbtToWordExport, false);  
 addButton(pnlUpButtons, jbtToExcelExport, true);  
 }  
  
 */\*\*  
 \* Метод setUpButtonsPanel устанавливает нижнюю панель кнопок с дополнительной информацией.  
 \*/* private void setDownButtonsPanel() {  
 JButton jbtAboutAppInfo = getJbtAboutApp();  
 JButton jbtAboutAuthorInfo = getJbtAboutAuthor();  
 JButton jbtExit = getExitButton();  
  
 stylizeButtons(*BUTTON\_BACKGROUND*, jbtAboutAppInfo, jbtAboutAuthorInfo);  
 stylizeButtons(Color.*RED*, jbtExit);  
  
 pnlDownButtonsPanel.setBackground(*PANEL\_BACKGROUND*);  
 pnlDownButtonsPanel.setBorder(*getTitledBorder*("Дополнительные сведения и действия"));  
 addButton(pnlDownButtonsPanel, jbtAboutAppInfo, false);  
 addButton(pnlDownButtonsPanel, jbtAboutAuthorInfo, false);  
 addButton(pnlDownButtonsPanel, jbtExit, true);  
 }  
  
 */\*\*  
 \* Возвращает объект TitledBorder с настроенным шрифтом и стилем.  
 \*  
 \* @param title Текст границы панели  
 \* @return объект TitledBorder с настроенным шрифтом и стилем  
 \*/* private static TitledBorder getTitledBorder(String title) {  
 TitledBorder pnlUpTitledBorder = BorderFactory.*createTitledBorder*(  
 BorderFactory.*createLineBorder*(Color.*lightGray*),  
 title,  
 TitledBorder.*CENTER*,  
 TitledBorder.*DEFAULT\_POSITION*);  
 pnlUpTitledBorder.setTitleFont(new Font("Montserrat", Font.*ITALIC*, 12));  
 return pnlUpTitledBorder;  
 }  
  
 */\*\*  
 \* Метод addButton добавляет кнопку на панель.  
 \*  
 \* @param pnlLayout Панель, которая будет содержать кнопки  
 \* @param button Кнопка.  
 \* @param isEndButton Флаг, указывающий, является ли кнопка последней на панели.  
 \*/* private void addButton(JPanel pnlLayout, JButton button, boolean isEndButton) {  
 pnlLayout.add(button);  
 if (isEndButton) return;  
 pnlLayout.add(Box.*createHorizontalGlue*());  
 }  
  
 */\*\*  
 \* Метод stylizeLabels задает стиль для надписей.  
 \*  
 \* @param foreground Цвет переднего плана.  
 \* @param labels Надписи.  
 \*/* private void stylizeLabels(Color foreground, JLabel... labels) {  
 for (JLabel label : labels) {  
 label.setFont(new Font("Montserrat", Font.*BOLD* | Font.*ITALIC*, 24));  
 label.setForeground(foreground);  
 }  
 }  
  
 */\*\*  
 \* Метод stylizeButtons задает стиль для кнопок.  
 \*  
 \* @param background Цвет заднего плана.  
 \* @param jButtons Кнопки.  
 \*/* private void stylizeButtons(Color background, JButton... jButtons) {  
 for (JButton jButton : jButtons) {  
 jButton.setPreferredSize(*BUTTON\_PREFFERED\_SIZE*);  
 jButton.setForeground(Color.*WHITE*);  
 jButton.setBackground(background);  
 jButton.setFont(new Font("Montserrat", Font.*BOLD* | Font.*ITALIC*, 14));  
 jButton.setBorder(null);  
 jButton.setBorder(new EmptyBorder(10, 10, 10, 10));  
 }  
 }  
  
 */\*\*  
 \* Конструктор контроллера формы группы.  
 \*/* public void initDBConnection() {  
 addWindowListener(new WindowAdapter() {  
 */\*\*  
 \* Вызывается в процессе закрытия окна formView.  
 \*  
 \* @param e экземпляр WindowEvent  
 \*/* @Override  
 public void windowClosing(WindowEvent e) {  
 try {  
 if (connection != null && !connection.isClosed()) {  
 sqLiteConnectionProvider.closeConnection();  
 }  
 } catch (SQLException exception) {  
 exception.printStackTrace();  
 }  
 }  
 });  
 initButtonsListeners();  
 }  
  
 */\*\*  
 \* Возвращает провайдера соединения SQLite.  
 \*  
 \* @return Провайдер соединения SQLite.  
 \*/* private SQLiteConnectionProvider getSqLiteConnectionProvider() {  
 SQLiteConnectionProvider sqLiteConnectionProvider = new SQLiteConnectionProvider();  
 connection = sqLiteConnectionProvider.getConnection();  
 *studentDAO* = new StudentDAO(connection);  
 return sqLiteConnectionProvider;  
 }  
  
 */\*\*  
 \* Инициализирует слушателей событий для кнопок на View.  
 \*/* private void initButtonsListeners() {  
 ActionListener actionListenerShowStudentsTable = this::actionShowStudentsTablePerformed;  
 ActionListener actionListenerExportToWord = e -> actionExportToWordPerformed();  
 ActionListener actionListenerExportToExcel = e -> actionExportToExcelPerformed();  
 jbtShowStudentsTable.addActionListener(actionListenerShowStudentsTable);  
 getJbtToWordExport().addActionListener(actionListenerExportToWord);  
 getJbtToExcelExport().addActionListener(actionListenerExportToExcel);  
 }  
  
 */\*\*  
 \* Метод, содержащий логику добавления таблицы,  
 \* осуществляемое соответствующей кнопкой.  
 \*  
 \* @param event Экземпляр сообщения о произошедшем событии  
 \*/* private void actionShowStudentsTablePerformed(ActionEvent event) {  
 if (pnlContentLayout.getComponentCount() == 0) {  
 Object[] columnsNames = Student.StudentsJTableModelInfo.*getTableColumnsNamesWithoutGroup*();  
 int columnsNumber = columnsNames.length;  
 Object[][] studentsData = SQLiteDBHelper.*getStudentsTableData*(connection, groupNumber, columnsNumber);  
 studentsTable = *getCustomLightJTableWithActionColumn*(studentsData, columnsNames);  
 studentsTable.setCustomBooleanIntegerRenderers(Student.StudentsJTableModelInfo.*getBooleanColumnsIndexes*(),  
 Student.StudentsJTableModelInfo.*getIntegerColumnsIndexes*());  
 initAttributesPanel(columnsNames, studentsTable.getColumnModel());  
 studentsTable.addActionColumn(getTableActionEvents());  
 if (studentsData.length == 0) {  
 initJTableInput(studentsTable);  
 }  
 JScrollPane tableScrollPane = new JScrollPane(studentsTable);  
 pnlContentLayout.add(tableScrollPane);  
 *addTablePopupMenu*(studentsTable);  
 revalidate();  
 repaint();  
 initTablePopupMenu(studentsTable);  
 }  
 }  
  
 */\*\*  
 \* Инициализирует панель атрибутов.  
 \*  
 \* @param columnNames имена столбцов  
 \* @param tableColumnModel модель столбцов таблицы  
 \*/* private void initAttributesPanel(Object[] columnNames, TableColumnModel tableColumnModel) {  
 if (pnlInnerAttributes.getComponentCount() != 0) return;  
 for (int i = 0, columnNamesLength = columnNames.length; i < columnNamesLength; i++) {  
 PillButton columnPillButton = *getPillButton*(columnNames[i]);  
 columnPillButton.setLblFont(new Font("Montserrat", Font.*ITALIC*, 10));  
 columnPillButton.setPreferredSize(StudentsFrame.*PILLS\_PREFFERED\_SIZE*);  
 columnPillButton.setMaximumSize(StudentsFrame.*PILLS\_PREFFERED\_SIZE*);  
 pnlInnerAttributes.add(columnPillButton);  
  
 TableColumn columnToHide = tableColumnModel.getColumn(i);  
 columnPillButton.addMouseListener(new MouseAdapter() {  
 */\*\*  
 \* Вызывается при щелчке мыши на кнопке.  
 \*  
 \* @param e экземпляр MouseEvent  
 \*/* @Override  
 public void mouseClicked(MouseEvent e) {  
 int normalWidth = getWidth() / columnNames.length;  
 int widthToSet = columnPillButton.getPillActivated() ? normalWidth : 0;  
 columnToHide.setMinWidth(widthToSet);  
 columnToHide.setMaxWidth(widthToSet);  
 columnToHide.setWidth(widthToSet);  
 columnToHide.setPreferredWidth(widthToSet);  
 }  
 });  
 }  
 }  
  
 */\*\*  
 \* Инициализирует всплывающее меню для таблицы студентов.  
 \*  
 \* @param studentsTable таблица студентов  
 \*/* private void initTablePopupMenu(CustomLightJTableWithActionColumn studentsTable) {  
 *popupMenu* = new JPopupMenu();  
  
 JMenuItem addMenuItem = new JMenuItem("Добавить строку");  
 addMenuItem.setAccelerator(KeyStroke.*getKeyStroke*(KeyEvent.*VK\_A*, InputEvent.*CTRL\_DOWN\_MASK*));  
  
 JMenuItem deleteMenuItem = new JMenuItem("Удалить студента");  
 deleteMenuItem.setAccelerator(KeyStroke.*getKeyStroke*(KeyEvent.*VK\_D*, InputEvent.*CTRL\_DOWN\_MASK*));  
  
 JMenuItem updateMenuItem = new JMenuItem("Обновить таблицу");  
 updateMenuItem.setFocusable(true);  
 updateMenuItem.setAccelerator(KeyStroke.*getKeyStroke*(KeyEvent.*VK\_U*, InputEvent.*CTRL\_DOWN\_MASK*));  
  
 addPopupMenuItems(studentsTable, addMenuItem, deleteMenuItem, updateMenuItem);  
 *popupMenu*.add(addMenuItem);  
 *popupMenu*.add(deleteMenuItem);  
 *popupMenu*.add(updateMenuItem);  
 }  
  
 */\*\*  
 \* Добавляет элементы контекстного меню в таблицу студентов.  
 \*  
 \* @param studentsTable таблица студентов  
 \* @param addMenuItem элемент меню "Добавить строку"  
 \* @param deleteMenuItem элемент меню "Удалить студента"  
 \* @param updateMenuItem элемент меню "Обновить таблицу"  
 \*/* private void addPopupMenuItems(CustomLightJTableWithActionColumn studentsTable, JMenuItem addMenuItem,  
 JMenuItem deleteMenuItem, JMenuItem updateMenuItem) {  
 addMenuItem.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 int rowIndex = studentsTable.getSelectedRow();  
 if (rowIndex < 0) {  
 if (*isEmptyTable*(studentsTable)) {  
 initJTableInput(studentsTable);  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Выберите строку таблицы, после " +  
 "которой" + " необходимо добавить студента");  
 }  
 return;  
 }  
 DefaultTableModel tableModel = (DefaultTableModel) studentsTable.getModel();  
 Object previousStudentID = 0;  
 previousStudentID = tableModel.getValueAt(rowIndex, 0);  
 *addRowToStudentsTable*(rowIndex, (int) previousStudentID + 1, tableModel, false);  
 setStudentsNum(String.*valueOf*((Integer.*parseInt*(studentsNum) + 1)));  
  
 }  
 });  
  
 deleteMenuItem.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 if (*isEmptyTable*(studentsTable)) return;  
 int[] selectedRowsIndexes = studentsTable.getSelectedRows();  
 if (selectedRowsIndexes.length == 0) {  
 JOptionPane.*showMessageDialog*(null, "Выберите студента, которого необходимо удалить");  
 return;  
 }  
 DefaultTableModel tableModel = (DefaultTableModel) studentsTable.getModel();  
 for (int rowIndex : selectedRowsIndexes) {  
 String studentID = tableModel.getValueAt(rowIndex, 0).toString();  
 *studentDAO*.deleteStudent(studentID);  
 tableModel.removeRow(rowIndex);  
 setStudentsNum(String.*valueOf*((Integer.*parseInt*(studentsNum) - 1)));  
 }  
 if (studentsTable.getModel().getRowCount() == 0) {  
 *addRowToStudentsTable*(0, 1, (DefaultTableModel) studentsTable.getModel(), true);  
 }  
 }  
 });  
  
 updateMenuItem.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 jbtShowStudentsTable.doClick();  
 }  
 });  
 }  
  
 */\*\*  
 \* Возвращает события действий таблицы.  
 \*  
 \* @return события действий таблицы  
 \*/* private TableActionCellEvent getTableActionEvents() {  
 return new TableActionCellEvents();  
 }  
  
 */\*\*  
 \* Возвращает модель студента из строки данных.  
 \*  
 \* @param rowData строка данных  
 \* @return модель студента  
 \*/* private Student getStudentDBModelFromRow(Object[] rowData) {  
 return new Student((int) rowData[0], groupNumber, rowData[1].toString(), rowData[2].toString(), rowData[3].toString(), (boolean) rowData[4], rowData[5].toString() == null ? "" : rowData[5].toString(), rowData[6].toString() == null ? "" : rowData[6].toString(), (boolean) rowData[7], rowData[8].toString());  
 }  
  
 */\*\*  
 \* Инициализирует ввод таблицы студентов.  
 \*  
 \* @param jTableStudentsList таблица студентов  
 \*/* private void initJTableInput(JTable jTableStudentsList) {  
 DefaultTableModel defaultTableModel = (DefaultTableModel) jTableStudentsList.getModel();  
 *addRowToStudentsTable*(0, 0, defaultTableModel, true);  
 setStudentsNum("1");  
 }  
  
 */\*\*  
 \* Метод для выполнения экспорта в файл.  
 \*  
 \* @param fileType тип файла для экспорта (WORD или EXCEL)  
 \*/* private void actionExportPerformed(FilePathChooserDialog.FileType fileType) {  
 TableExporter exporter;  
 if (fileType == FilePathChooserDialog.FileType.*WORD*) {  
 exporter = new TableExporterToWord();  
 } else if (fileType == FilePathChooserDialog.FileType.*EXCEL*) {  
 exporter = new TableExporterToExcel();  
 } else {  
 throw new IllegalArgumentException("Неподдерживаемый тип файла");  
 }  
  
 String filepath = FilePathChooserDialog.*createFile*(fileType);  
 exporter.export(studentsTable, filepath);  
 }  
  
 */\*\*  
 \* Метод для выполнения экспорта в файл Word.  
 \*/* private void actionExportToWordPerformed() {  
 if (studentsTable != null) {  
 actionExportPerformed(FilePathChooserDialog.FileType.*WORD*);  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Отобразите таблицу, которую необходимо вывести.");  
 }  
 }  
  
 */\*\*  
 \* Метод для выполнения экспорта в файл Excel.  
 \*/* private void actionExportToExcelPerformed() {  
 if (studentsTable != null) {  
 actionExportPerformed(FilePathChooserDialog.FileType.*EXCEL*);  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Отобразите таблицу, которую необходимо вывести.");  
 }  
 }  
  
 */\*\*  
 \* Метод setStudentsNum устанавливает количество студентов.  
 \*  
 \* @param studentsNum Количество студентов.  
 \*/* public void setStudentsNum(String studentsNum) {  
 lblStudentsNumValue.setText(studentsNum);  
 }  
  
 */\*\*  
 \* Возвращает кнопку jbtToWordExport.  
 \*  
 \* @return кнопка jbtToWordExport  
 \*/* public JButton getJbtToWordExport() {  
 return jbtToWordExport;  
 }  
  
 */\*\*  
 \* Возвращает кнопку jbtToExcelExport.  
 \*  
 \* @return кнопка jbtToExcelExport  
 \*/* public JButton getJbtToExcelExport() {  
 return jbtToExcelExport;  
 }  
  
 */\*\*  
 \* Класс, реализующий события действий таблицы.  
 \*/* private class TableActionCellEvents implements TableActionCellEvent {  
  
 */\*\*  
 \* Добавляет строку в таблицу.  
 \*  
 \* @param rowIndex индекс строки  
 \* @param jTable таблица  
 \*/* @Override  
 public void onAddRow(int rowIndex, JTable jTable) {  
 try {  
 DefaultTableModel tableModel = (DefaultTableModel) jTable.getModel();  
 Object newRowStudentID = (int) tableModel.getValueAt(rowIndex, 0) + 1;  
 *addRowToStudentsTable*(rowIndex, newRowStudentID, tableModel, false);  
 setStudentsNum(String.*valueOf*((Integer.*parseInt*(studentsNum) + 1)));  
 } catch (ClassCastException e) {  
 JOptionPane.*showMessageDialog*(null, "Извините, что-то пошло не так");  
 }  
 }  
  
 */\*\*  
 \* Удаляет строку из таблицы.  
 \*  
 \* @param rowIndex индекс строки  
 \* @param jTable таблица  
 \*/* @Override  
 public void onDelete(int rowIndex, JTable jTable) {  
 DefaultTableModel tableModel = (DefaultTableModel) jTable.getModel();  
 String studentID = tableModel.getValueAt(rowIndex, 0).toString();  
 *studentDAO*.deleteStudent(studentID);  
 int rowCount = tableModel.getRowCount();  
 if (rowIndex >= 0 && rowIndex < rowCount) {  
 tableModel.removeRow(rowIndex);  
 }  
 if (jTable.getModel().getRowCount() == 0) {  
 *addRowToStudentsTable*(0, 1, (DefaultTableModel) jTable.getModel(), true);  
 }  
 }  
  
 */\*\*  
 \* Обновляет базу данных.  
 \*  
 \* @param rowIndex индекс строки  
 \* @param jTable таблица  
 \*/* @Override  
 public void onUpdateDB(int rowIndex, JTable jTable) {  
 DefaultTableModel tableModel = (DefaultTableModel) jTable.getModel();  
 int columnCount = tableModel.getColumnCount();  
 Object[] rowData = new Object[columnCount];  
 for (int i = 0; i < columnCount; i++) {  
 rowData[i] = tableModel.getValueAt(rowIndex, i);  
 }  
 SQLiteDBHelper.StudentsDataValidator.ValidityConstants validityResult = SQLiteDBHelper.StudentsDataValidator.*validateData*(rowData, connection, groupNumber);  
 if (validityResult == SQLiteDBHelper.StudentsDataValidator.ValidityConstants.*NOT\_VALID\_VALUES*) {  
 return;  
 } else if (validityResult == SQLiteDBHelper.StudentsDataValidator.ValidityConstants.*VALID\_ROW*) {  
 *studentDAO*.addStudentToDB(getStudentDBModelFromRow(rowData));  
 setStudentsNum(String.*valueOf*(Integer.*parseInt*(studentsNum) + 1));  
 } else if (validityResult == SQLiteDBHelper.StudentsDataValidator.ValidityConstants.*UPDATE\_ROW*) {  
 *studentDAO*.updateStudentInDB(getStudentDBModelFromRow(rowData));  
 }  
 pnlContentLayout.removeAll();  
 jbtShowStudentsTable.doClick();  
 }  
 }  
}

Класс **AboutAuthorFrame**

package Frames;  
  
import javax.swing.\*;  
import javax.swing.border.Border;  
import java.awt.\*;  
  
*/\*\*  
 \* Класс AboutAuthorFrame представляет собой окно с информацией об авторе.  
 \* Окно содержит фотографию автора, его имя, адрес электронной почты и номер группы.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.1  
 \*/*public class AboutAuthorFrame extends JFrame {  
  
 */\*\*  
 \* Минимальный размер окна.  
 \*/* private final Dimension MINIMUM\_FRAME\_SIZE = new Dimension(500, 800);  
  
 */\*\*  
 \* Цвет фона.  
 \*/* private final Color BACKGROUND\_COLOR = new Color(243, 243, 243);  
  
 */\*\*  
 \* Пустая граница.  
 \*/* private final Border EMPTY\_BORDER = BorderFactory.*createEmptyBorder*(10, 0, 10, 0);  
  
 */\*\*  
 \* Граница кнопки "Выход".  
 \*/* private final Border MATTE\_BORDER = BorderFactory.*createMatteBorder*(10, 10, 5, 10, BACKGROUND\_COLOR);  
  
 */\*\*  
 \* Метка для фотографии автора.  
 \*/* private JLabel lblPhoto;  
  
 */\*\*  
 \* Метка для имени автора.  
 \*/* private JLabel lblName;  
  
 */\*\*  
 \* Метка для адреса электронной почты автора.  
 \*/* private JLabel lblEmail;  
  
 */\*\*  
 \* Метка для номера группы автора.  
 \*/* private JLabel lblGroup;  
  
 */\*\*  
 \* Метка для текста "Автор".  
 \*/* private JLabel lblAuthor;  
  
 */\*\*  
 \* Кнопка для выхода из окна.  
 \*/* private JButton jbtnExit;  
  
 */\*\*  
 \* Конструктор класса AboutAuthorFrame.  
 \* Инициализирует окно и его компоненты.  
 \*/* public AboutAuthorFrame() {  
 setMinimumSize(MINIMUM\_FRAME\_SIZE);  
 ImageIcon icon = new ImageIcon("assets/AuthorFrameIcon.png");  
 setIconImage(icon.getImage());  
  
 setTitle("Об Авторе");  
 setSize(MINIMUM\_FRAME\_SIZE.getSize().width, MINIMUM\_FRAME\_SIZE.getSize().height);  
 setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
 setLayout(new BorderLayout());  
 setBackground(BACKGROUND\_COLOR);  
  
 createComponents();  
 addComponents();  
  
 setVisible(true);  
 }  
  
 */\*\*  
 \* Метод для создания компонентов.  
 \*/* private void createComponents() {  
 lblPhoto = new JLabel();  
 setIcon(lblPhoto);  
 lblPhoto.setHorizontalAlignment(SwingConstants.*CENTER*);  
 lblPhoto.setBorder(EMPTY\_BORDER);  
  
 lblAuthor = new JLabel("Автор");  
 configureLabel(lblAuthor);  
  
 lblName = new JLabel("студент группы 10702221");  
 configureLabel(lblName);  
  
 lblEmail = new JLabel("Будчанин Вадим Александрович");  
 configureLabel(lblEmail);  
  
 lblGroup = new JLabel("vadik.trolla@gmail.com");  
 configureLabel(lblGroup);  
  
 jbtnExit = new JButton("Назад");  
 jbtnExit.setBorder(MATTE\_BORDER);  
  
 jbtnExit.addActionListener(e -> {  
 dispose();  
 });  
 }  
  
 */\*\*  
 \* Метод для добавления компонентов.  
 \*/* private void addComponents() {  
 *// Создаем панель для текстовых меток* JPanel textPanel = new JPanel();  
  
 textPanel.setLayout(new BoxLayout(textPanel, BoxLayout.*Y\_AXIS*));  
 textPanel.setBackground(BACKGROUND\_COLOR);  
  
 textPanel.add(lblAuthor);  
 textPanel.add(lblName);  
 textPanel.add(lblEmail);  
 textPanel.add(lblGroup);  
  
 add(lblPhoto, BorderLayout.*NORTH*);  
  
 add(textPanel, BorderLayout.*CENTER*);  
  
 add(jbtnExit, BorderLayout.*SOUTH*);  
 getContentPane().setBackground(BACKGROUND\_COLOR);  
 }  
  
 */\*\*  
 \* Метод для настройки метки.  
 \* @param label Метка для настройки.  
 \*/* private void configureLabel(JLabel label) {  
 label.setFont(new Font("Montserrat", Font.*BOLD*, 14));  
 label.setHorizontalAlignment(SwingConstants.*CENTER*);  
 label.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 }  
  
 */\*\*  
 \* Метод для установки иконки метки из файла.  
 \*  
 \* @param label Метка, для которой устанавливается иконка.  
 \*/* private void setIcon(JLabel label) {  
  
 ImageIcon icon = new ImageIcon("assets/AuthorPhoto.jpg");  
  
 Image image = icon.getImage();  
 *// Масштабируем изображение с сохранением качества* Image newimg = image.getScaledInstance(  
 MINIMUM\_FRAME\_SIZE.getSize().width - 20,  
 MINIMUM\_FRAME\_SIZE.getSize().height - 180,  
 java.awt.Image.*SCALE\_SMOOTH*);  
  
 icon = new ImageIcon(newimg);  
  
 label.setIcon(icon);  
 }  
}

Класс **AboutAppFrame**

package Frames;  
  
import javax.swing.\*;  
import javax.swing.text.SimpleAttributeSet;  
import javax.swing.text.StyleConstants;  
import javax.swing.text.StyledDocument;  
import java.awt.\*;  
  
*/\*\*  
 \* Класс AboutAppFrame представляет собой окно с информацией о приложении.  
 \* Окно информация с титульного листа курсовой.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.1  
 \*/*public class AboutAppFrame extends JFrame {  
 */\*\*  
 \* Версия приложения.  
 \*/* private static final String APP\_VERSION = "1.0.0";  
  
 */\*\*  
 \* Минимальный размер окна.  
 \*/* private final Dimension MINIMUM\_FRAME\_SIZE = new Dimension(600, 600);  
  
 */\*\*  
 \* Цвет фона.  
 \*/* private final Color BACKGROUND\_COLOR = new Color(243,243,243);  
  
 */\*\*  
 \* Описание приложения.  
 \*/* private final String APP\_DESCRIPTION = """  
 Приложение "Студенческая группа" предоставляет следующие возможности:  
   
 1. Создание группы: Пользователь может создать новую группу, указав уникальное имя группы.  
 2. Добавление студентов в группу: Пользователь может добавлять новых студентов в каждую группу, указывая их данные, такие как имя, фамилия, номер зачетной книжки и т.д.  
 3. Редактирование данных студента: Пользователь может изменять данные студента, выбрав студента и внося необходимые изменения.  
 4. Удаление студента из группы: Пользователь может удалить студента из группы, выбрав студента и нажав кнопку удаления.  
 5. Экспорт данных группы: Пользователь может экспортировать данные группы в формате Excel или Word. Это может быть полезно для создания отчетов или анализа данных.  
   
 Эта программа предоставляет удобный и эффективный способ управления студенческими группами. Она обеспечивает гибкость и контроль над данными студентов, позволяя пользователю легко добавлять, редактировать и удалять студентов, а также экспортировать данные для дальнейшего анализа или отчетности.""";  
  
 */\*\*  
 \* Конструктор класса AboutAppFrame.  
 \*/* public AboutAppFrame() {  
 setupFrame();  
 addComponents();  
 setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);  
 setVisible(true);  
 }  
  
 */\*\*  
 \* Метод для настройки основных параметров окна.  
 \*/* private void setupFrame() {  
 ImageIcon icon = new ImageIcon("assets/AboutAppIcon.png");  
 setBackground(BACKGROUND\_COLOR);  
 getContentPane().setBackground(BACKGROUND\_COLOR);  
 setMinimumSize(MINIMUM\_FRAME\_SIZE);  
 setSize(MINIMUM\_FRAME\_SIZE);  
 setTitle("О программе");  
 setIconImage(icon.getImage());  
 setLayout(new BoxLayout(getContentPane(), BoxLayout.Y\_AXIS));  
 }  
  
 */\*\*  
 \* Метод для добавления компонентов на панель окна.  
 \*/* private void addComponents() {  
 JPanel pnlBottom = new JPanel(new FlowLayout());  
 pnlBottom.add(createVersionLabel());  
 pnlBottom.add(createExitButton());  
 add(createTopPanel());  
 add(pnlBottom);  
 }  
  
 */\*\*  
 \* Метод для создания верхней панели окна.  
 \* @return JPanel объект.  
 \*/* private JPanel createTopPanel() {  
 JPanel topPanel = new JPanel();  
 topPanel.setBackground(BACKGROUND\_COLOR);  
 topPanel.setLayout(new BoxLayout(topPanel, BoxLayout.X\_AXIS));  
  
 topPanel.add(createImageLabel());  
 topPanel.add(Box.createHorizontalStrut(20));  
 topPanel.add(createTextArea());  
  
 return topPanel;  
 }  
  
 */\*\*  
 \* Метод для создания метки изображения.  
 \* @return JLabel объект.  
 \*/* private JLabel createImageLabel() {  
 ImageIcon imageIcon = new ImageIcon("assets/AppDiagram.png");  
 Image image = imageIcon.getImage();  
 Image newimg = image.getScaledInstance(500/2, 650/2, java.awt.Image.SCALE\_SMOOTH);  
 imageIcon = new ImageIcon(newimg);  
  
 return new JLabel(imageIcon);  
 }  
  
 */\*\*  
 \* Метод для создания текстовой области.  
 \* @return JTextArea объект.  
 \*/* private JTextPane createTextArea() {  
 JTextPane textPane = new JTextPane();  
 textPane.setBackground(BACKGROUND\_COLOR);  
 textPane.setText(APP\_DESCRIPTION);  
 textPane.setEditable(false);  
  
 *// Center align the text* StyledDocument doc = textPane.getStyledDocument();  
 SimpleAttributeSet center = new SimpleAttributeSet();  
 StyleConstants.setAlignment(center, StyleConstants.ALIGN\_CENTER);  
 doc.setParagraphAttributes(0, doc.getLength(), center, false);  
  
 return textPane;  
 }  
  
 */\*\*  
 \* Метод для создания метки версии.  
 \* @return JLabel объект.  
 \*/* private JLabel createVersionLabel() {  
 return new JLabel("Версия программы: " + APP\_VERSION);  
 }  
  
 */\*\*  
 \* Метод для создания кнопки выхода.  
 \* @return JButton объект.  
 \*/* private JButton createExitButton() {  
 JButton exitButton = new JButton("Выход");  
 exitButton.addActionListener(e -> dispose());  
  
 return exitButton;  
 }  
  
 public static void main(String[] args) {  
 AboutAppFrame frame = new AboutAppFrame();  
 frame.setVisible(true);  
 }  
}

Класс **GroupDAO**

package Database.DAOS;  
  
import Frames.models.Group;  
  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Класс GroupDAO предоставляет методы для взаимодействия с таблицей групп в базе данных.  
 \* <p>  
 \* Автор: Будчанин В.А.  
 \* Версия: 1.1  
 \*/*public class GroupDAO {  
 */\*\*  
 \* Соединение с базой данных  
 \*/* private final Connection connectionDB;  
  
 */\*\*  
 \* Конструктор класса GroupDAO  
 \*  
 \* @param connectionDB объект Connection для подключения к базе данных  
 \*/* public GroupDAO(Connection connectionDB) {  
 this.connectionDB = connectionDB;  
 }  
  
 */\*\*  
 \* Метод для добавления группы в базу данных  
 \*  
 \* @param groupNumber номер группы  
 \* @param courseNumber номер курса  
 \* @param headmanFullName полное имя старосты  
 \*/* public void addGroup(String groupNumber, int courseNumber, String headmanFullName) {  
 String query = "INSERT INTO StudentGroups (groupNumber, course, headmanFullName) VALUES (?, ?, ?)";  
 try (PreparedStatement insertStatement = connectionDB.prepareStatement(query)) {  
 insertStatement.setString(1, groupNumber);  
 insertStatement.setInt(2, courseNumber);  
 insertStatement.setString(3, headmanFullName);  
 insertStatement.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Метод для редактирования данных группы в базе данных  
 \*  
 \* @param oldGroupNumber старый номер группы  
 \* @param newGroupNumber новый номер группы  
 \* @param courseNumber номер курса  
 \* @param headmanFullName полное имя старосты  
 \*/* public void editGroupData(String oldGroupNumber, String newGroupNumber, int courseNumber, String headmanFullName) {  
 int stringEquals = oldGroupNumber.compareTo(newGroupNumber);  
 try {  
 if (stringEquals == 0) {  
 String updateQuery = "UPDATE StudentGroups SET course = ?, headmanFullName = ? WHERE groupNumber = ?";  
 PreparedStatement statement = connectionDB.prepareStatement(updateQuery);  
 statement.setInt(1, courseNumber);  
 statement.setString(2, headmanFullName);  
 statement.setString(3, newGroupNumber);  
 statement.executeUpdate();  
 } else {  
 deleteGroup(oldGroupNumber);  
 addGroup(newGroupNumber, courseNumber, headmanFullName);  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Метод для получения списка всех групп из базы данных  
 \*  
 \* @return список объектов GroupViews, представляющих группы  
 \*/* public List<Group> getAllGroups() {  
 List<Group> groupList = new ArrayList<>();  
 String query = "SELECT \* FROM StudentGroups";  
 try (PreparedStatement statement = connectionDB.prepareStatement(query)) {  
 ResultSet resultSet = statement.executeQuery();  
 while (resultSet.next()) {  
 String groupNumber = resultSet.getString("groupNumber");  
 int course = resultSet.getInt("course");  
 String headmanFullName = resultSet.getString("headmanFullName");  
 Group group = new Group(groupNumber, course, headmanFullName);  
 groupList.add(group);  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return groupList;  
 }  
  
 */\*\*  
 \* Метод для удаления группы из базы данных  
 \*  
 \* @param groupNumber номер группы  
 \*/* public void deleteGroup(String groupNumber) {  
 String deleteGroup = "DELETE FROM StudentGroups WHERE groupNumber = ?;";  
 try (PreparedStatement statement = connectionDB.prepareStatement(deleteGroup)) {  
 statement.setString(1, groupNumber);  
 statement.executeUpdate();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Метод для проверки, пуста ли таблица "StudentGroups" в базе данных.  
 \*  
 \* @return true, если таблица пуста, иначе false.  
 \*/* public boolean isTableEmpty() {  
 boolean isEmpty = false;  
 try {  
 Statement statement = connectionDB.createStatement();  
 ResultSet resultSet = statement.executeQuery("SELECT COUNT(\*) FROM StudentGroups");  
 if (resultSet.next()) {  
 int count = resultSet.getInt(1);  
 isEmpty = (count == 0);  
 }  
 resultSet.close();  
 statement.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return isEmpty;  
 }  
}

Класс **StudentDAO**

package Database.DAOS;  
  
import Frames.models.Student;  
import org.sqlite .SQLiteErrorCode;  
  
import javax.swing.\*;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Класс StudentDAO предоставляет методы для взаимодействия с таблицей студентов в базе данных.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.0  
 \*/*public class StudentDAO {  
 */\*\*  
 \* Соединение с базой данных  
 \*/* private final Connection connection;  
  
 */\*\*  
 \* Конструктор класса StudentDAO.  
 \*  
 \* @param connection объект Connection для подключения к базе данных  
 \*/* public StudentDAO(Connection connection) {  
 this.connection = connection;  
 }  
  
 */\*\*  
 \* Метод для получения списка всех студентов из конкретной группы из базы данных.  
 \*  
 \* @param groupNumber номер группы  
 \* @return список объектов Student, представляющих студентов  
 \*/* public List<Student> getAllGroupStudents(String groupNumber) {  
 List<Student> students = new ArrayList<>();  
 String query = "SELECT \* FROM Students WHERE groupNumber = ?";  
 try (PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setString(1, groupNumber);  
 try (ResultSet resultSet = statement.executeQuery()) {  
 while (resultSet.next()) {  
 int studentID = resultSet.getInt("studentID");  
 String firstName = resultSet.getString("firstname");  
 String surname = resultSet.getString("surname");  
 String middleName = resultSet.getString("middleName");  
 boolean isPayer = resultSet.getBoolean("isPayer");  
 String currentAddress = resultSet.getString("currentAddress");  
 String homeAddress = resultSet.getString("homeAddress");  
 boolean isLocal = resultSet.getBoolean("isLocal");  
 String phoneNumber = resultSet.getString("phoneNumber");  
 Student student = new Student(studentID, groupNumber, firstName, surname, middleName, isPayer,  
 homeAddress, currentAddress, isLocal, phoneNumber);  
 students.add(student);  
 }  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return students;  
 }  
  
 */\*\*  
 \* Метод для добавления студента в таблицу.  
 \*  
 \* @param student модель студента, содержащая данные для добавления  
 \*/* public void addStudentToDB(Student student) {  
 String query = "INSERT INTO Students (studentID, groupNumber, firstname, surname, middleName, isPayer, homeAddress, currentAddress, isLocal, phoneNumber) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";  
 try (PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setInt(1, student.getStudentID());  
 statement.setString(2, student.getGroupNumber());  
 statement.setString(3, student.getFirstName());  
 statement.setString(4, student.getSurname());  
 statement.setString(5, student.getMiddleName());  
 statement.setBoolean(6, student.getIsPayer());  
 statement.setString(7, student.getHomeAddress());  
 statement.setString(8, student.getCurrentAddress());  
 statement.setBoolean(9, student.getIsLocal());  
 statement.setString(10, student.getPhoneNumber());  
 statement.executeUpdate();  
 } catch (SQLException e) {  
 String message;  
 if (e.getErrorCode() == SQLiteErrorCode.*SQLITE\_CONSTRAINT*.code){  
 message = "Вероятно, уже есть студент с таким номером (номером телефона или номером-идентификатором)" +  
 ", удалите данного студента или перепроверьте номер.";  
 }  
 else {  
 message = "Произошли непредвиденные проблемы при добавлении записи в базу данных";  
 }  
 JOptionPane.*showMessageDialog*(null, message);  
 }  
 }  
  
 */\*\*  
 \* Обновляет информацию о студенте в базе данных.  
 \*  
 \* @param student Объект студента, содержащий обновленную информацию о студенте.  
 \*/* public void updateStudentInDB(Student student) {  
 String query = "UPDATE Students SET groupNumber = ?, firstname = ?, surname = ?, middleName = ?, isPayer = ?, currentAddress = ?, homeAddress = ?, isLocal = ?, phoneNumber = ? WHERE studentID = ?";  
 try (PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setString(1, student.getGroupNumber());  
 statement.setString(2, student.getFirstName());  
 statement.setString(3, student.getSurname());  
 statement.setString(4, student.getMiddleName());  
 statement.setBoolean(5, student.getIsPayer());  
 statement.setString(6, student.getCurrentAddress());  
 statement.setString(7, student.getHomeAddress());  
 statement.setBoolean(8, student.getIsLocal());  
 statement.setString(9, student.getPhoneNumber());  
 statement.setInt(10, student.getStudentID());  
 statement.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Проверяет наличие студента с указанным идентификатором.  
 \*  
 \* @param studentID идентификатор студента для проверки  
 \* @param groupNumber номер группы  
 \* @return true, если студент с указанным идентификатором существует; в противном случае - false  
 \*/* public boolean isStudentExists(String studentID, String groupNumber) {  
 String query = "SELECT COUNT(\*) AS count FROM Students WHERE studentID = ? AND groupNumber = ?";  
 try (PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setString(1, studentID);  
 statement.setString(2, groupNumber);  
 try (ResultSet resultSet = statement.executeQuery()) {  
 if (resultSet.next()) {  
 int count = resultSet.getInt("count");  
 return count > 0;  
 }  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return false;  
 }  
  
 */\*\*  
 \* Возвращает количество студентов в определенной группе.  
 \*  
 \* @param groupNumber номер группы, для которой нужно получить количество студентов  
 \* @return количество студентов в группе  
 \*/* public int getCountOfGroupStudents(String groupNumber) {  
 int count = 0;  
 String query = "SELECT COUNT(\*) as count FROM Students WHERE groupNumber = ?";  
 try (PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setString(1, groupNumber);  
 try (ResultSet resultSet = statement.executeQuery()) {  
 if (resultSet.next()) {  
 count = resultSet.getInt("count");  
 }  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return count;  
 }  
  
  
 */\*\*  
 \* Метод для удаления студента из базы данных.  
 \*  
 \* @param studentID идентификатор студента  
 \*/* public void deleteStudent(String studentID) {  
 String query = "DELETE FROM Students WHERE studentID = ?";  
 try (PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setString(1, studentID);  
 statement.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
}

Класс **SQLiteConnectionProvider**

package Database.Managers;  
  
import org.sqlite.SQLiteConfig;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
  
*/\*\*  
 \* Провайдер подключения к базе данных SQLite.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.0  
 \*/*public class SQLiteConnectionProvider {  
 */\*\*  
 \* URL базы данных.  
 \*/* private static final String *DB\_URL* = "jdbc:sqlite:D:\\3COURSE\\Java\\CourseWork\\CourseWorkApp\\src\\Database\\DatabaseFiles\\StudentGroupDB.sqlite";  
  
 */\*\*  
 \* Соединение с базой данных.  
 \*/* private Connection connection;  
  
 */\*\*  
 \* Создает новый провайдер подключения к базе данных SQLite.  
 \*/* public SQLiteConnectionProvider() {  
 try {  
 Class.*forName*("org.sqlite.JDBC");  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Возвращает подключение к базе данных.  
 \*  
 \* @return объект Connection для подключения к базе данных  
 \*/* public Connection getConnection() {  
 if (connection == null) {  
 try {  
 SQLiteConfig config = new SQLiteConfig();  
 config.enforceForeignKeys(true);  
 connection = DriverManager.*getConnection*(*DB\_URL*, config.toProperties());  
 System.*out*.println("Connected to SQLite database.");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 return connection;  
 }  
  
 */\*\*  
 \* Закрывает подключение к базе данных.  
 \*/* public void closeConnection() {  
 if (connection != null) {  
 try {  
 connection.close();  
 System.*out*.println("Connection to SQLite database closed.");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

Интерфейс **SQLiteDBHelper**

package Database.Managers;  
  
import Database.DAOS.StudentDAO;  
import Frames.models.Student;  
  
import javax.swing.\*;  
import java.sql.Connection;  
import java.util.List;  
  
*/\*\*  
 \* Класс-по, осуществляющий периферийные действия между базой данных и контроллером.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.1  
 \*/*public class SQLiteDBHelper {  
  
 */\*\*  
 \* Получает данные таблицы студентов.  
 \*  
 \* @param connection соединение с базой данных  
 \* @param groupNumber номер группы  
 \* @param columnsNum количество столбцов  
 \* @return двумерный массив объектов с данными таблицы студентов  
 \*/* public static Object[][] getStudentsTableData(Connection connection, String groupNumber, int columnsNum) {  
 StudentDAO studentDAO = new StudentDAO(connection);  
 List<Student> studentList = studentDAO.getAllGroupStudents(groupNumber);  
 Object[][] tableData = new Object[studentList.size()][columnsNum];  
 int rowIndex = 0;  
 for (Student student : studentList) {  
 Object[] rowElements = new Object[columnsNum];  
 rowElements[0] = student.getStudentID();  
 rowElements[1] = student.getFirstName();  
 rowElements[2] = student.getSurname();  
 rowElements[3] = student.getMiddleName();  
 rowElements[4] = student.getIsPayer();  
 rowElements[5] = student.getHomeAddress();  
 rowElements[6] = student.getCurrentAddress();  
 rowElements[7] = student.getIsLocal();  
 rowElements[8] = student.getPhoneNumber();  
 tableData[rowIndex] = rowElements;  
 rowIndex++;  
 }  
 return tableData;  
 }  
  
 */\*\*  
 \* Класс, осуществляющий валидацию вводимых значений в строку о студента  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.3  
 \*/* public static class StudentsDataValidator {  
  
 */\*\*  
 \* Валидирует данные строки.  
 \*  
 \* @param dataRow данные строки  
 \* @param connection соединение с базой данных  
 \* @param groupNumber номер группы  
 \* @return константа, указывающая на результат валидации  
 \*/* public static ValidityConstants validateData(Object[] dataRow, Connection connection, String groupNumber) {  
 ValidityConstants validityResult = *validateStudentID*(dataRow[0], connection, groupNumber);  
 if (validityResult == ValidityConstants.*NOT\_VALID\_VALUES* || !*isValidString*(dataRow[1]) || !*isValidString*(dataRow[2])  
 || !*isValidString*(dataRow[3]) || !*isValidBoolean*(dataRow[4])) {  
 JOptionPane.*showMessageDialog*(null, "Не были введены обязательные данные.");  
 return ValidityConstants.*NOT\_VALID\_VALUES*;  
 }  
 return validityResult;  
 }  
  
 */\*\*  
 \* Проверяет, является ли значение допустимым идентификатором студента.  
 \*  
 \* @param value значение, которое необходимо проверить  
 \* @param connection соединение с базой данных  
 \* @param groupNumber номер группы  
 \* @return константа, указывающая на результат проверки  
 \*/* private static ValidityConstants validateStudentID(Object value, Connection connection, String groupNumber) {  
 if (value instanceof Integer) {  
 StudentDAO studentDAO = new StudentDAO(connection);  
 int studentID = (int) value;  
 boolean isValid = studentID >= 0 && !studentDAO.isStudentExists(String.*valueOf*(studentID), groupNumber);  
 if (!isValid) {  
 int choice = JOptionPane.*showOptionDialog*(null, "Студент с таким же номером существует. Что нужно сделать",  
 "Диалоговое окно с выбором", JOptionPane.*DEFAULT\_OPTION*, JOptionPane.*QUESTION\_MESSAGE*,  
 null, new Object[]{"Обновить запись", "Отменить добавление новой записи"}, "Отменить добавление новой записи");  
 return switch (choice) {  
 */\*  
 Обновить запись.  
 \*/* case 0 -> ValidityConstants.*UPDATE\_ROW*;  
 */\*  
 Закрыть редактирование.  
 \*/* case 1 -> ValidityConstants.*CLOSE\_EDIT*;  
 */\*  
 Недопустимые значения.  
 \*/* default -> ValidityConstants.*NOT\_VALID\_VALUES*;  
 };  
 } else {  
 */\*  
 Допустимая запись.  
 \*/* return ValidityConstants.*VALID\_ROW*;  
 }  
 }  
 */\*  
 Недопустимые значения.  
 \*/* return ValidityConstants.*NOT\_VALID\_VALUES*;  
 }  
  
 */\*\*  
 \* Проверяет, является ли значение допустимой строкой.  
 \*  
 \* @param value значение, которое необходимо проверить  
 \* @return true, если значение является допустимой строкой, иначе false  
 \*/* private static boolean isValidString(Object value) {  
 return value instanceof String && !((String) value).isEmpty();  
 }  
  
 */\*\*  
 \* Проверяет, является ли значение допустимым булевым значением.  
 \*  
 \* @param value значение, которое необходимо проверить  
 \* @return true, если значение является допустимым булевым значением, иначе false  
 \*/* private static boolean isValidBoolean(Object value) {  
 return value instanceof Boolean;  
 }  
  
 */\*\*  
 \* Константы, указывающие на результат проверки.  
 \*/* public enum ValidityConstants {  
 */\*\*  
 \* Допустимая запись.  
 \*/  
 VALID\_ROW*,  
 */\*\*  
 \* Обновить запись.  
 \*/  
 UPDATE\_ROW*,  
 */\*\*  
 \* Закрыть редактирование.  
 \*/  
 CLOSE\_EDIT*,  
 */\*\*  
 \* Недопустимые значения.  
 \*/  
 NOT\_VALID\_VALUES*,  
 }  
 }  
}

Класс **Group**

package Frames.models;  
  
*/\*\*  
 \* Модель группы.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.0  
 \*/*public class Group {  
 */\*\*  
 \* Номер группы.  
 \*/* private String groupNumber;  
  
 */\*\*  
 \* Номер курса.  
 \*/* private int courseNumber;  
  
 */\*\*  
 \* Полное имя старосты.  
 \*/* private String headmanFullName;  
  
 */\*\*  
 \* Создает новую модель группы.  
 \*  
 \* @param groupNumber номер группы  
 \* @param courseNumber номер курса  
 \* @param headmanFullName полное имя старосты  
 \*/* public Group(String groupNumber, int courseNumber, String headmanFullName) {  
 this.groupNumber = groupNumber;  
 this.courseNumber = courseNumber;  
 this.headmanFullName = headmanFullName;  
 }  
  
 */\*\*  
 \* Возвращает номер группы.  
 \*  
 \* @return номер группы  
 \*/* public String getGroupNumber() {  
 return groupNumber;  
 }  
  
 */\*\*  
 \* Возвращает номер курса.  
 \*  
 \* @return номер курса  
 \*/* public int getCourseNumber() {  
 return courseNumber;  
 }  
  
 */\*\*  
 \* Возвращает полное имя старосты.  
 \*  
 \* @return полное имя старосты  
 \*/* public String getHeadmanFullName() {  
 return headmanFullName;  
 }  
  
}

Класс **Student**

package Frames.models;  
  
*/\*\*  
 \* Модель студента, т.е. сведения о его номере, группе, ФИО, местах жительства, а также о том, местный ли и обучается ли  
 \* на платной основе  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.1  
 \*/*public class Student {  
 */\*\*  
 \* Идентификатор студента.  
 \*/* private final int studentID;  
  
 */\*\*  
 \* Номер группы студента.  
 \*/* private final String groupNumber;  
  
 */\*\*  
 \* Фамилия студента.  
 \*/* private final String surname;  
  
 */\*\*  
 \* Отчество студента.  
 \*/* private final String middleName;  
  
 */\*\*  
 \* Имя студента.  
 \*/* private final String firstName;  
  
 */\*\*  
 \* Показывает, является ли студент плательщиком.  
 \*/* private final boolean isPayer;  
  
 */\*\*  
 \* Домашний адрес студента.  
 \*/* private String homaAddress;  
  
 */\*\*  
 \* Текущий адрес студента.  
 \*/* private String currentAddress;  
  
 */\*\*  
 \* Показывает, является ли студент местным.  
 \*/* private boolean isLocal;  
  
 */\*\*  
 \* Номер телефона студента.  
 \*/* private String phoneNumber;  
  
  
 */\*\*  
 \* Конструктор класса Student  
 \*  
 \* @param magazineStudentNumber номер студента в журнале  
 \* @param groupNumber номер группы  
 \* @param surname фамилия студента  
 \* @param middleName отчество студента  
 \* @param firstName имя студента  
 \* @param isPayer является ли студент плательщиком  
 \*/* public Student(int magazineStudentNumber, String groupNumber, String surname, String middleName, String firstName, boolean isPayer) {  
 studentID = magazineStudentNumber;  
 this.groupNumber = groupNumber;  
 this.surname = surname;  
 this.middleName = middleName;  
 this.firstName = firstName;  
 this.isPayer = isPayer;  
 this.homaAddress = null;  
 this.currentAddress = null;  
 this.isLocal = false;  
 }  
  
 */\*\*  
 \* Конструктор класса Student  
 \*  
 \* @param magazineStudentNumber номер студента в журнале  
 \* @param groupNumber номер группы  
 \* @param surname фамилия студента  
 \* @param middleName отчество студента  
 \* @param firstName имя студента  
 \* @param isPayer является ли студент плательщиком  
 \* @param homaAddress домашний адрес студента  
 \* @param currentAddress текущий адрес студента  
 \* @param isLocal является ли студент местным  
 \* @param phoneNumber номер телефона студента  
 \*/* public Student(int magazineStudentNumber, String groupNumber, String firstName, String surname, String middleName, boolean isPayer, String homaAddress, String currentAddress, boolean isLocal, String phoneNumber) {  
 this(magazineStudentNumber, groupNumber, surname, middleName, firstName, isPayer);  
 this.homaAddress = homaAddress;  
 this.currentAddress = currentAddress;  
 this.isLocal = isLocal;  
 this.phoneNumber = phoneNumber;  
 }  
  
 */\*\*  
 \* Метод для получения номера студента в журнале  
 \*  
 \* @return номер студента в журнале  
 \*/* public int getStudentID() {  
 return studentID;  
 }  
  
 */\*\*  
 \* Метод для получения номера группы  
 \*  
 \* @return номер группы  
 \*/* public String getGroupNumber() {  
 return groupNumber;  
 }  
  
 */\*\*  
 \* Метод для получения фамилии студента  
 \*  
 \* @return фамилия студента  
 \*/* public String getSurname() {  
 return surname;  
 }  
  
 */\*\*  
 \* Метод для получения отчества студента  
 \*  
 \* @return отчество студента  
 \*/* public String getMiddleName() {  
 return middleName;  
 }  
  
 */\*\*  
 \* Метод для получения имени студента  
 \*  
 \* @return имя студента  
 \*/* public String getFirstName() {  
 return firstName;  
 }  
  
 */\*\*  
 \* Метод для получения информации о том, является ли студент плательщиком  
 \*  
 \* @return true, если студент является плательщиком, иначе false  
 \*/* public boolean getIsPayer() {  
 return isPayer;  
 }  
  
 */\*\*  
 \* Метод для получения домашнего адреса студента  
 \*  
 \* @return домашний адрес студента  
 \*/* public String getHomeAddress() {  
 return homaAddress;  
 }  
  
 */\*\*  
 \* Метод для получения текущего адреса студента  
 \*  
 \* @return текущий адрес студента  
 \*/* public String getCurrentAddress() {  
 return currentAddress;  
 }  
  
 */\*\*  
 \* Метод для получения информации о том, является ли студент местным  
 \*  
 \* @return true, если студент является местным, иначе false  
 \*/* public boolean getIsLocal() {  
 return isLocal;  
 }  
  
 */\*\*  
 \* Получить номер телефона.  
 \* @return номер телефона  
 \*/* public String getPhoneNumber() {  
 return phoneNumber;  
 }  
  
 */\*\*  
 \* Этот класс представляет модель данных для отображения студентов в JTable.  
 \*  
 \* @author Будчанин В.А.  
 \* @version 1.0  
 \*/* public static class StudentsJTableModelInfo {  
  
 */\*\*  
 \* Возвращает представление таблицы в виде DefaultTableModel.  
 \*  
 \* @return DefaultTableModel, представляющая таблицу.  
 \*/* public static String[] getTableColumnsNamesWithoutGroup() {  
 return new String[]{  
 "НОМЕР СТУДЕНТА\*",  
 "ИМЯ\*",  
 "ФАМИЛИЯ\*",  
 "ОТЧЕСТВО\*",  
 "ПЛАТНИК\*",  
 "ДОМАШНИЙ АДРЕС",  
 "ТЕКУЩИЙ АДРЕС",  
 "МЕСТНЫЙ",  
 "ТЕЛЕФОН",  
 };  
 }  
  
 */\*\*  
 \* Этот метод возвращает индексы столбцов, которые содержат булевы значения.  
 \*  
 \* @return Массив индексов столбцов, содержащих булевы значения.  
 \*/* public static int[] getBooleanColumnsIndexes(){  
 int isPayerColumnNumber = 4;  
 int isLocalColumnNumber = 7;  
 return new int[] {isPayerColumnNumber, isLocalColumnNumber};  
 }  
  
 */\*\*  
 \* Этот метод возвращает индексы столбцов, которые содержат целочисленные значения.  
 \*  
 \* @return Массив индексов столбцов, содержащих целочисленные значения.  
 \*/* public static int[] getIntegerColumnsIndexes(){  
 return new int[] { 0 };  
 }  
 }  
}