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 **«Заселение студентов в общежитие»**

|  |  |
| --- | --- |
| Выполнила: | студентка группы 10702220  Кишкурно Мария Вячеславовна |
| Руководитель: | к.ф.-м.н., доц.  Сидорик Валерий Владимирович |

Минск 2022

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**

по дисциплине **«Программирование на языке Java»**

на тему **«Заселение студентов в общежитие»**

|  |  |
| --- | --- |
| Выполнила: | студентка группы 10702220  Кишкурно Мария Вячеславовна |
| Руководитель: | к.ф.-м.н., доц.  Сидорик Валерий Владимирович |

Минск 2022

**Содержание**

[ВВЕДЕНИЕ…………………………………………………………….…](#_heading=h.gjdgxs).5

[1. Постановка и описание задачи………………………………………...](#_heading=h.30j0zll)6

[2. Теоретически](#_heading=h.1fob9te)й раздел………………………………………………….7

[2.1 Язык программирования Java](#_heading=h.3znysh7)…………………………………..7

[2.2 Платформа JavaFX](#_heading=h.1t3h5sf)………………………………………………8

3. Математическая модель………………………………………………10

[4.](#_heading=h.4d34og8) Практический раздел………………………………………………….11

4.1 Описание структуры проекта………………………………….11

4.2 UML-диаграмма классов………………………………………12

4.3 Использование СУБД MySQL………………………………...13

[4.](#_heading=h.17dp8vu)4 [Реализация графических окон](#_heading=h.17dp8vu)………………………………....15

[4.5 Стартовое окно………………………………………………....](#_heading=h.3rdcrjn)15

[4.6 Главное окно…………………………………………………....](#_heading=h.26in1rg)16

4.7 Окно с информацией о комнате……………………………….18

[4.8 Окно с информацией об авторе……………………………….. 1](#_heading=h.lnxbz9)9

[4.9 Окно с информацией о программе……………………………1](#_heading=h.35nkun2)9

4.10 Утилита JavaDoc………………………………………………20

4.11 Сборка jar-файла………………………………………………20

5. Результаты работы……………………………………………………21

[ЗАКЛЮЧЕНИЕ…………………………………………………...……..](#_heading=h.z337ya)27

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ](#_heading=h.3j2qqm3)…………………….28

[ПРИЛОЖЕНИЕ](#_heading=h.1y810tw) 1………………………………………………………..29

# **ВВЕДЕНИЕ**

Каждый учебный год для иногородних студентов начинается с заселения в общежитие. Невозможно держать в голове огромный объём информации о студентах и их заселении в комнаты, поэтому в целях избежать потери данных и автоматизации заселения студентов был разработан данный курсовой проект.

Интерфейс программы позволяет производить ввод данных студента, осуществлять просмотр имеющихся данных о студентах, расформировывать студентов по комнатам и получать информацию о проживающих студентах в определённой комнате. Вся имеющаяся информация сохраняется и хранится в базе данных.

В данной записке приведено описание программы «Заселение студентов в общежитие», формирующей и использующей базу данных MySQL.

# **ПОСТАНОВКА И ОПИСАНИЕ ЗАДАЧИ**

Тема курсовой работы: разработать приложение для заселения студентов в общежитие 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 студенте, визуальное распределение студентов по комнатам в общежитии с возможностью сохранения данных о студентах и их заселении.

Исходные данные (информация о студенте): фамилия, имя, отчество, пол, курс, группа, номер телефона.

Выходные данные: список студентов, распределённых по комнатам.

Технологии, используемые в работе:

1. Операционная система Windows 10;
2. Язык программирования Java;
3. Платформа JavaFX, CУБД MySQL;
4. Среда разработки Intellij IDEA 2022.2.1.

# **ТЕОРЕТИЧЕКИЙ РАЗДЕЛ**

* 1. **Язык программирования Java**

Язык программирования Java уже много лет занимает топовые места в рейтинге самых популярных технологий.

Java — это язык программирования общего назначения, который следует парадигме объектно-ориентированного программирования. Java используется для настольных, сетевых, мобильных и корпоративных приложений.

Java можно найти везде. Это основной язык [разработки для Android](https://www.altexsoft.com/blog/engineering/top-20-tools-for-android-development/). Он используется в веб-приложениях, правительственных веб-сайтах и ​​технологиях обработки больших данных. Java подходит и для научных проектов, особенно в области обработки естественного языка. Язык Java преобладал и в программировании для мобильных устройств, задолго [до появления смартфонов](https://www.pocketgamer.biz/feature/10705/a-brief-history-of-mobile-games-2002-wake-up-and-smell-the-coffee/) — первые мобильные игры в начале 2000-х годов были написаны на Java.

Сферы использования Java:

1. **Приложения для Android**. Несмотря на [активный рост Kotlin](https://appdevelopermagazine.com/why-kotlin-language-use-is-skyrocketing/), Java по-прежнему остается де-факто основным языком Android-приложений. Таким образом, все разработчики Java очень легко могут стать Android-программистами.
2. **Программные продукты.**Помимо уже упомянутых Hadoop и Apache Storm, Java использовалась для создания Eclipse, OpenOffice, Gmail, Atlassian и других.
3. **Финансовые программы.** Java — [один из самых востребованных языков](https://news.efinancialcareers.com/uk-en/137065/the-six-hottest-programming-languages-to-know-in-banking-technology) в финансовой отрасли. Он используется для создания надежных, быстрых и простых веб-сайтов как на стороне сервера, так и на стороне клиента.
4. **Кассовые терминалы**. Многие компании используют Java для создания систем PoS, поскольку их создание обычно требует кроссплатформенности и обширного штата специалистов.
5. **Торговые системы.** На Java написана Murex, популярная программа управления банками для фронтальной и обратной связи.
6. **Программы для работы с большими данными.** Hadoop написан на Java. Scala, Kafka и Spark используют JVM. Кроме того, Java предоставляет доступ к множеству проверенных библиотек, инструментов отладки и мониторинга.
   1. **Платформа JavaFX**

JavaFX - это платформа на основе Java, которая используется для разработки настольных приложений, а также полнофункциональных интернет-приложений (RIA). Приложения, созданные на JavaFX, могут работать на нескольких платформах, включая Интернет, мобильные приложения.

JavaFX предоставляет богатый набор графических и мультимедийных API и использует современный графический процессор с помощью аппаратного ускорения графики. JavaFX также предоставляет интерфейсы, с помощью которых разработчики могут комбинировать графическую анимацию и управление пользовательским интерфейсом.

Архитектуру Java FX можно хорошо понять из приведенной ниже диаграммы (рис 1). API JavaFX работают поверх различных уровней. Под этими API-интерфейсами находится множество вспомогательных функций, на которых построен Java FX. Здесь он работает поверх квантового инструментария. Этот инструментарий работает с различными инструментами, поддерживающими этот инструментарий.

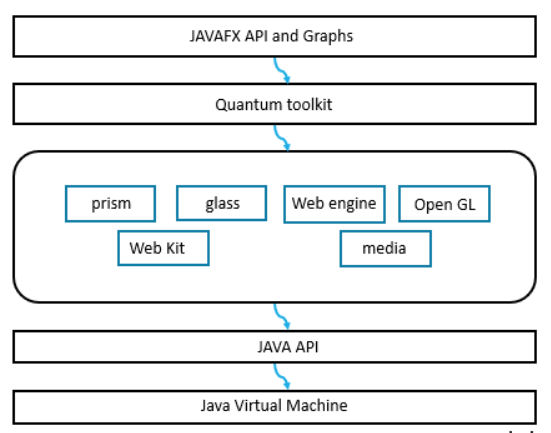


Рис. 1 – Архитектура JavaFX

В структуре JavaFX лежит три основных компонента: Stage, Scene, Node (рис.2)

Основой для создания графического интерфейса в JavaFX является класс javafx.stage.Stage. По сути он является контейнером, в который помещаются все остальные компоненты интерфейса. Его конкретная реализация зависит от платформы, на которой запускается приложение. Так, на ноутбуках и компьютерах это будет отдельное графическое окно, а на мобильных устройствах интерфейс может представлять весь экран устройства. Можно использовать множество объектов Stage, но один из них является основным. При запуске приложения основной объект Stage создается средой JavaFX и передается в метод start().

Класс javafx.scene.Scene представляет контейнер для всех графических элементов внутри объекта Stage в виде графа, который называется Scene Graph. Для установки сцены у объекта Stage вызывается метод setScene(), в который передается объект Scene.

Все узлы этого графа, то есть по сути все вложенные элементы должны представлять класс javafx.scene.Node, представляющий все графические элементы.

Но корневой узел этого графа должен представлять объект класса, который унаследован от javafx.scene.Parent. По сути Parent - это контейнер, который может содержать другие элементы и управляет вложенными узлами. При этом все конструкторы принимают в качестве первого параметра корневой узел. То есть при создании объекта Scene нам та или иначе придется установить и корневой узел.

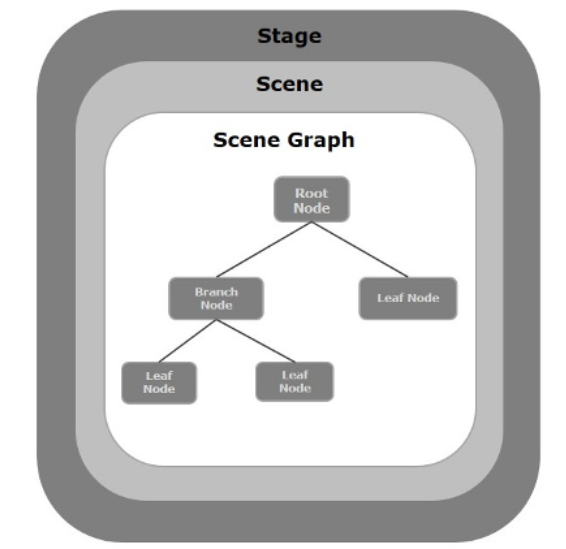


Рис. 2 – Структура JavaFX

# **МАТЕМАТИЧЕСКАЯ МОДЕЛЬ**

Приложение содержит:

1. *Окно входа в приложение.* В котором отображается начальная информация о проекте, 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 студенте, используя текстовые поля, добавив данные вручную, и распределить (заселить) его в определённую комнату, выбрав нужный этаж, нажав на ячейку комнаты и выбрав из существующего списка определённого студента.
3. *Окно с информацией об авторе.* С актуальной информацией об авторе программы, контактные данные.
4. *Окно с информацией о программе.* С отображением информации о версии приложения, описание функционала программы, предоставляемой интерфейсом.

Для того, чтобы заселить студента в комнату нужно знать его ФИО, пол, курс обучения, номер группы и номер телефона, данные будут передаваться путем ввода информации в текстовые поля в основном окне программы.

Студенты разного гендера не могут быть заселены в одну комнату. Студент не может быть заселён в общежитие дважды. В комнату можно заселить максимум 3 человека, а также можно просмотреть личную информацию о студентах, распределённых в ту или иную комнату. При попытке противоречить правилам заселения, данные функции будут недоступны.

# **ПРАКТИЧЕСКИЙ РАЗДЕЛ**

* 1. **Описание структуры проекта**

Приложение «Заселение студентов в общежитие» имеет следующую структуру (рис. 3).

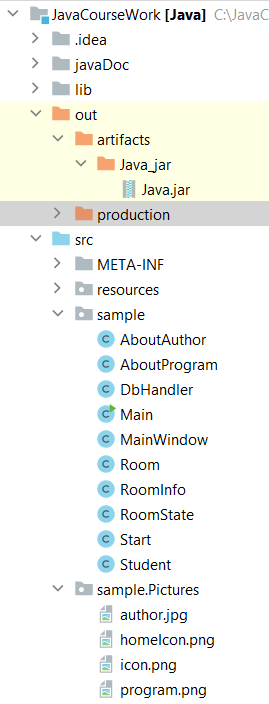
****

Рисунок 3 – Структура проекта

JavaCourseWork – папка со всеми файлами проекта;

Src – папка с файлами источниками .java;

javaDoc – папка с файлами JavaDoc;

Java.jar – исполняемый файл всего проекта.

* 1. **UML-диаграмма классов**

Унифицированный язык моделирования (UML) – это язык для специфицирования, визуализации, конструирования и документирования программных систем, а также бизнес-моделей и прочих не программных систем. UML представляет собой объединение инженерных приемов, которые ранее успешно использовались при моделировании больших и сложных систем. При решении сложных задач заблаговременное планирование и моделирование значительно упрощают программирование.

Ниже представлена UML диаграммы классов для приложения «Заселение студентов в общежитие» (рис.4)

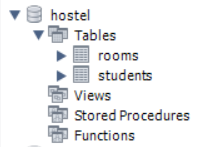


Рисунок 4 – UML-диаграмма классов

* 1. **Использование СУБД MySQL**

MySQL – это СУБД, поддерживающая SQL (универсальный язык программирования). Его поддерживают все системы управления базами данных.

Для удобства хранения данных, подключим к проекту базу данных, ведь мы не можем предугадать объём данных, с которыми придётся работать, а их потеря недопустима. Так как по задумке нам удобно иметь список всех студентов и список комнат, создадим соответственные таблицы: rooms и students и свяжем их (рис. 4.1)



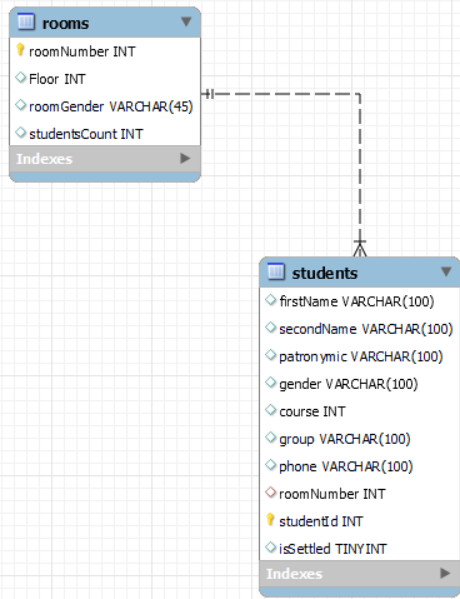


Рисунок 4.1 – Диаграмма связей таблиц

При каждом обращении к базе данных будет создаваться соединение (66 строка) и прописываться запрос (68 строка). В данном случае будет возвращаться список комнат на определённом этаже (рис. 4.2)

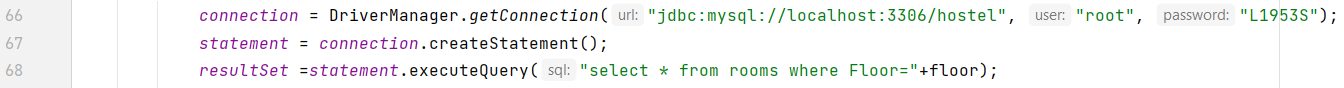


Рисунок 4.2 – Подключение БД и формирование запроса

При желании внести данные в базу данных нужно прописать соединение (122 строка) и соответствующий запрос (120 строка) на получение данных из Text Fields главного окна приложения (рис.4.3).

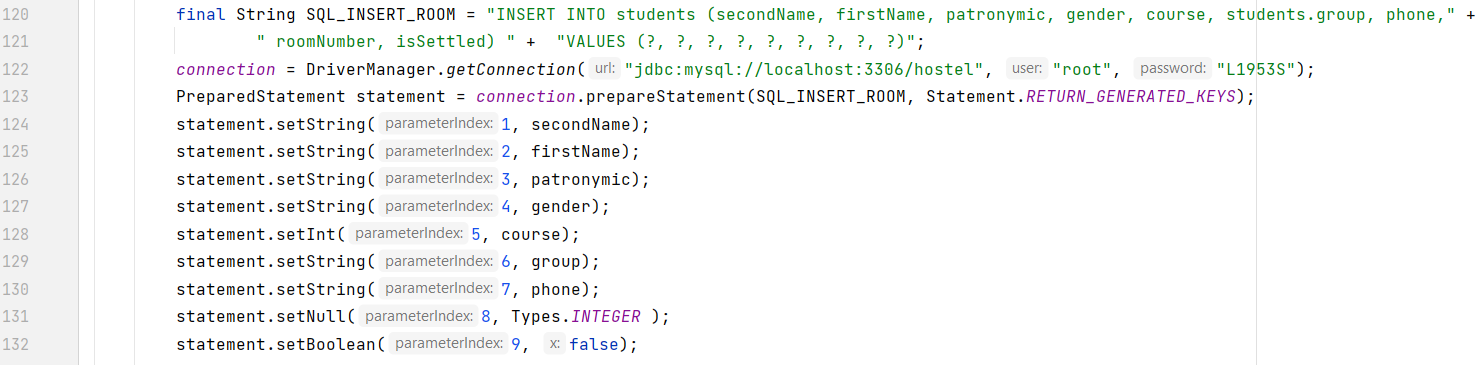
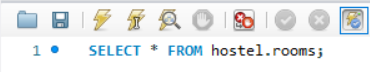


Рисунок 4.3 – Подключение БД и формирование запроса

То есть при каждом новом обращении к базе данных нужно прописать адрес соединения с базой данных и запрос, который требуется. Просмотреть состояния полей и повзаимодействовать с ними можно напрямую в базе данных (рис.4.4)



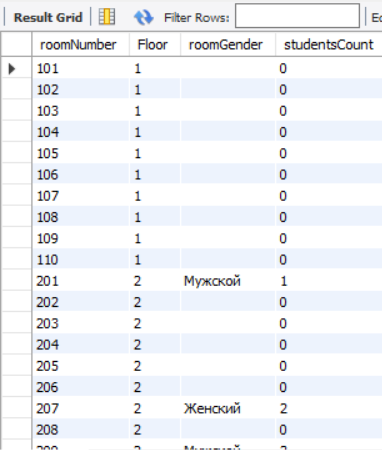


Рисунок 4.4 – Состояния полей таблицы rooms

Базу данных предпочитают использовать, так как с помощью запросов доступно: создание и изменение таблиц, настройка связи между ними, внесение и изменение информации, настройка доступа для разных абонентов, поиск необходимых сведений по всевозможным параметрам.

* 1. **Реализация графических окон**

Проект представлен блок-схемой или, как ее по-другому называют, бизнес-логикой переключения окон проекта, взаимосвязанных между собой:

* Splash Screen – стартовое окно приложения,
* Main Window – основное окно для работы с приложением,
* About program – окно, предоставляющее информацию о программе
* About Author – окно, предоставляющее информацию об авторе (рис. 5):

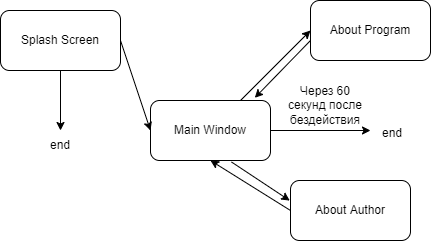


Рисунок 5 – Бизнес-логика переключения графических окон

* 1. **Стартовое окно**

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)

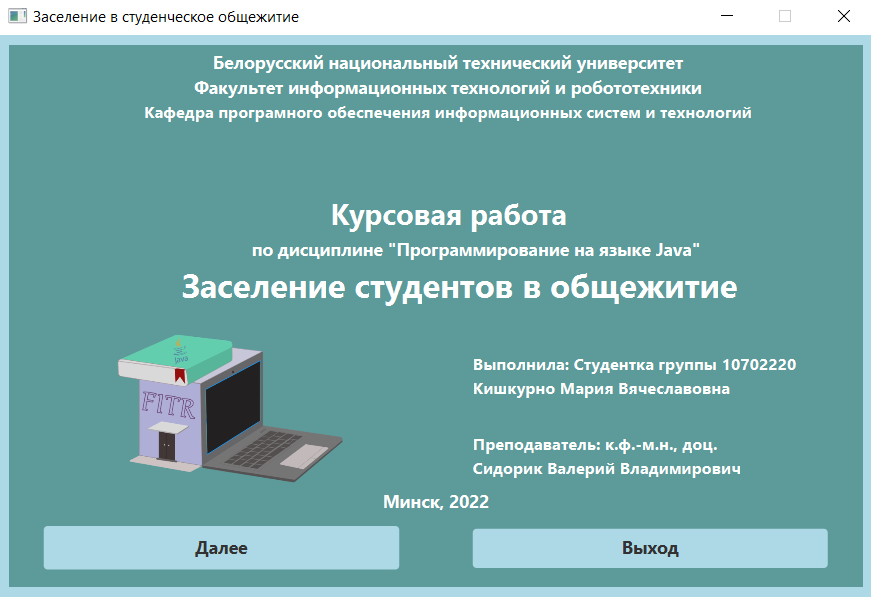
****

Рисунок 6 – Стартовое окно

* 1. **Главное окно**

При нажатии на кнопку «Далее» в основном окне перед пользователем открывается рабочее окно, содержащее меню с пунктами «Файл», имеющего всплывающую вкладку «Выйти», и «Информация», имеющее всплывающие вкладки «О программе», «Об авторе» (рис.7). При бездействии пользователя в течение 60 секунд, программа закрывается.

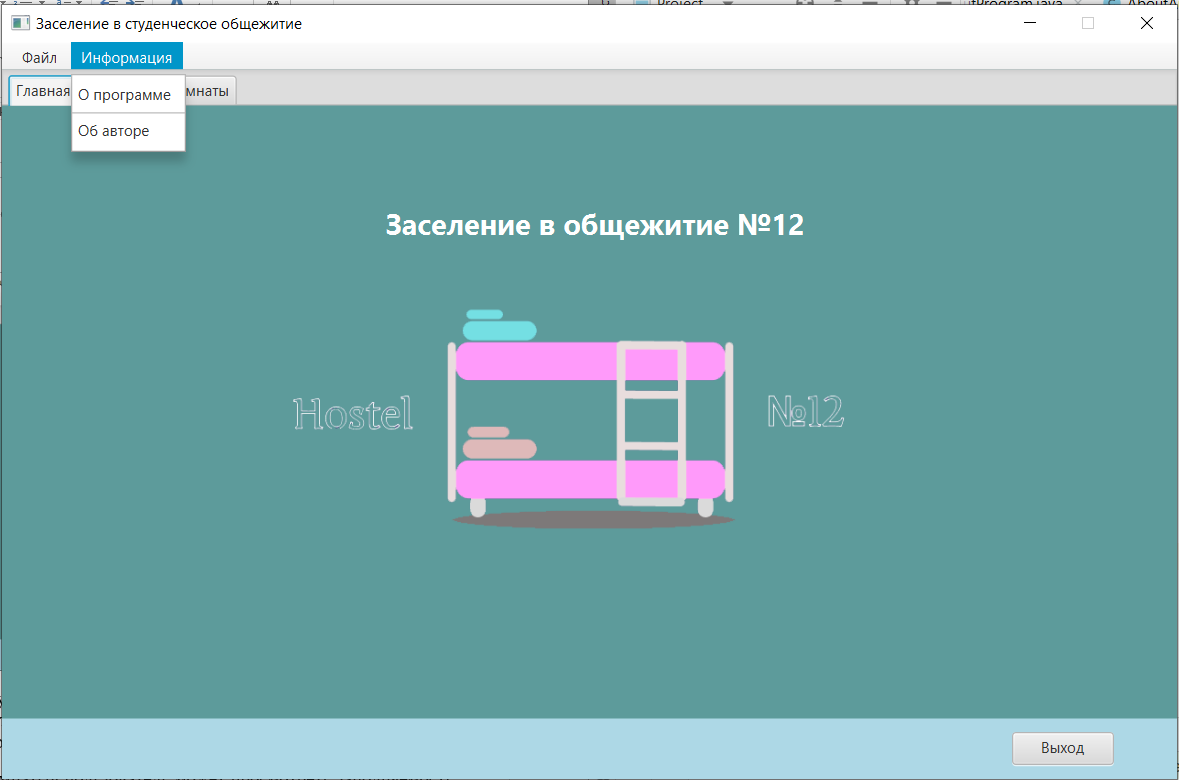


Рисунок 7 – Меню

С помощью элемента управления TabPane созданы вкладки «Главная», «Студенты», «Комнаты», с помощью которых можно переключаться к различным этапам. Изначально открывается вкладка под названием «Главная» (рис.8)

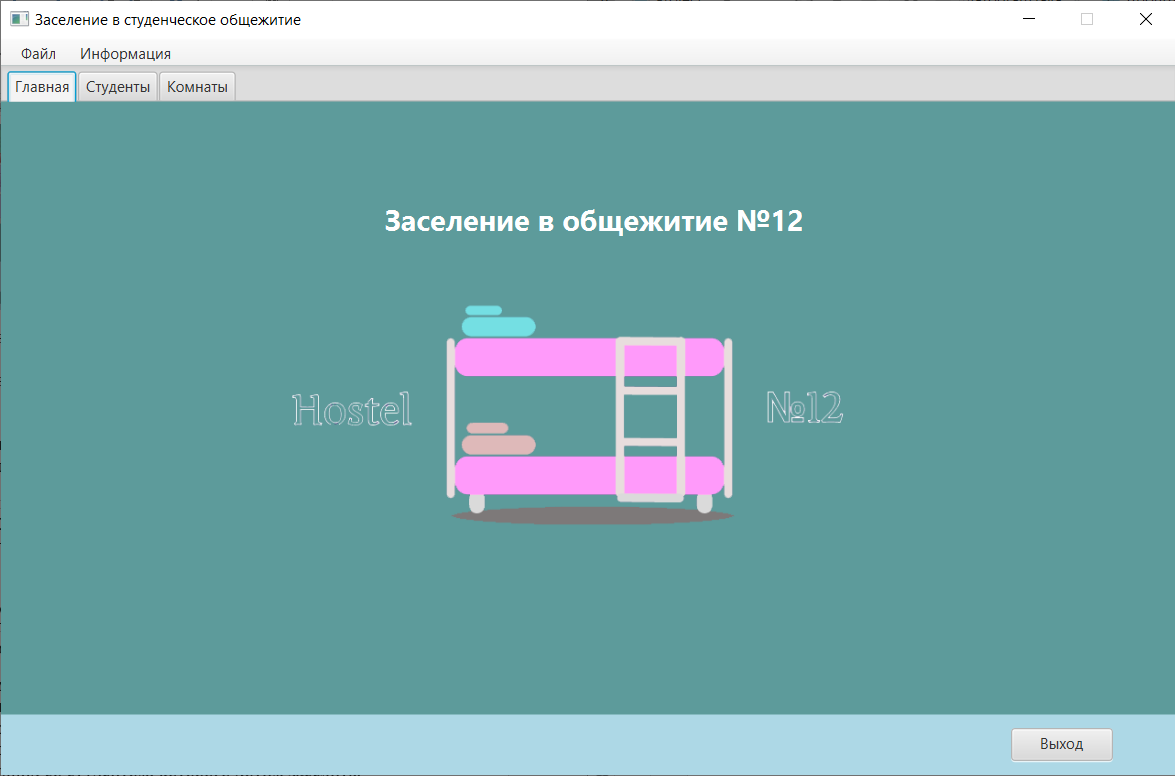


Рисунок 8 – Главное окно

Во вкладке «Студенты» пользователь имеет возможность ввести данные о новом студенте и добавить их в базу данных, а также просмотреть список студентов, претендующих на место в общежитии, который автоматически выгружается из базы данных (рис.9).

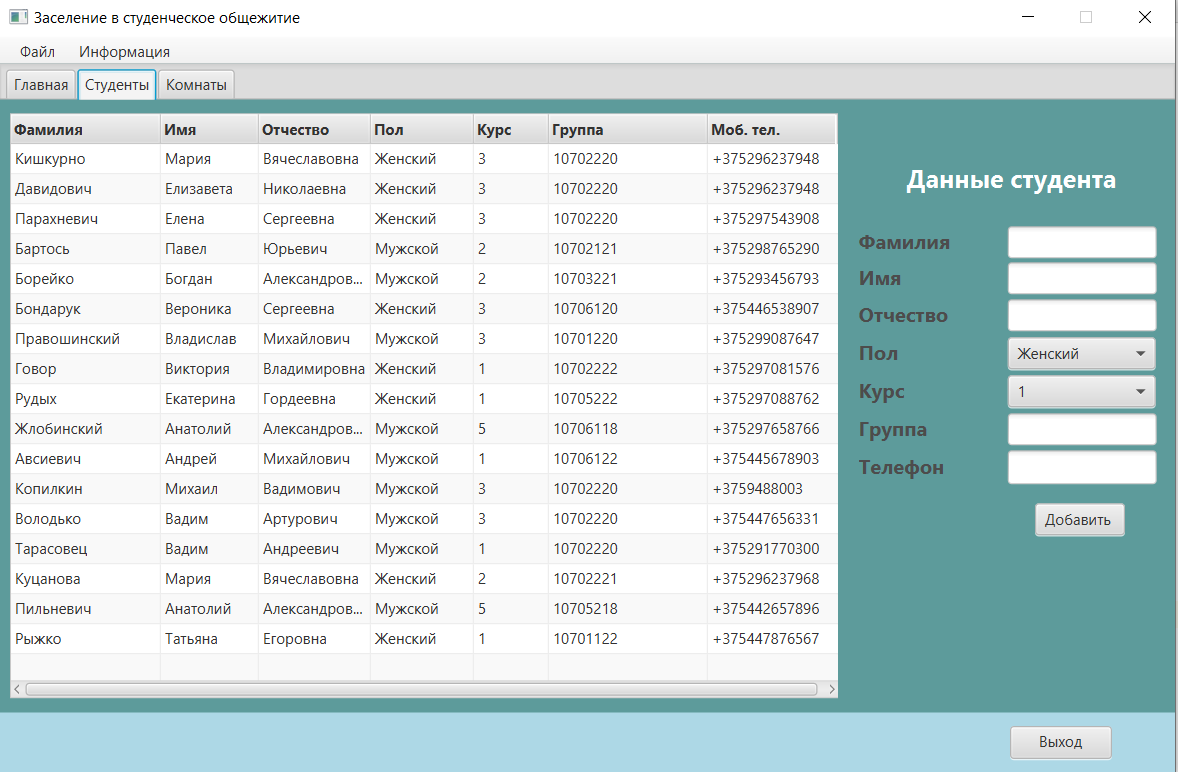


Рисунок 9 – Вкладка «Студенты»

Во вкладке «Комнаты» пользователь может просмотреть заполняемость комнат на выбор на пяти в общежитии, при нажатии на ячейку комнаты получить более детальную информацию о жильцах каждой комнаты. Можно так же выбрать и заселить студента в комнату, при условии наличия в комнате места и совпадения полов проживающих. После заселения информацию о комнатах можно обновить, нажав на кнопку «Обновить» (рис.10)

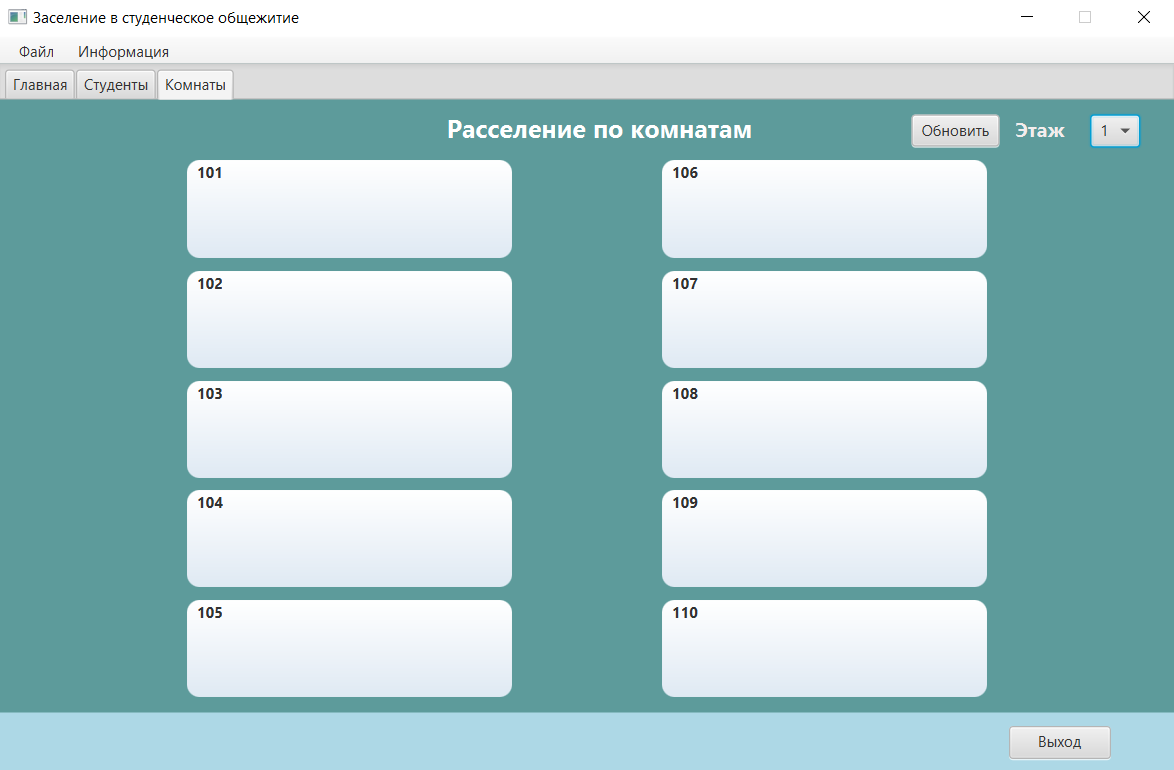


Рисунок 10 – Вкладка «Комнаты»

* 1. **Окно с информацией о комнате**

Окно, в котором представлена информация о заселённых студентах и реализована возможность выбора и заселения студента (рис.11).

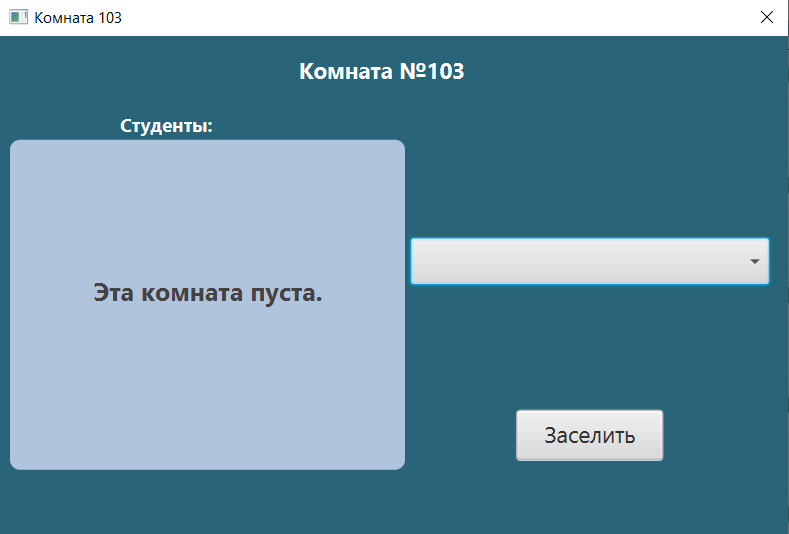


Рисунок 10 – Вкладка «Комнаты»

* 1. **Окно с информацией об авторе**

Окно «Об авторе» (рис. 11) содержит основную информацию об авторе проекта: номер группы, имя, фамилию, отчество, фотографию автора, а также адрес его электронной почты.

Если данной информационное окно открыто, то пользователь не может взаимодействовать с другими окнами. Чтобы вернуться обратно к меню, нужно нажать на кнопку «Назад», расположенную в окне.



Рисунок 11 – Окно «Об авторе»

* 1. **Окно с информацией о программе**

Окно «О программе» (рис. 12) содержит основную информацию о возможностях программы и версию приложения. Если данной информационное окно открыто, то пользователь не может взаимодействовать с другими окнами. Чтобы вернуться обратно к меню, нужно нажать на кнопку «Назад», расположенную в окне.



Рисунок 11 – Окно «О программе»

* 1. **Утилита JavaDoc**

С помощью утилиты javadoc, входящей в состав JDK, комментарий документации можно извлекать и помещать в НТМL файл. Утилита javadoc позволяет вставлять HTML тэги и использовать специальные ярлыки (дескрипторы) документирования. НТМL тэги заголовков не используют, чтобы не нарушать стиль файла, сформированного утилитой (рис. 12)

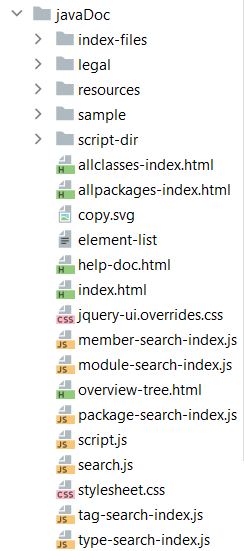
****

Рисунок 12 – Сгенерированные утилитой JavaDoc файлы

* 1. **Сборка jar-файла**

Весь проект объединяем в единый jar-файл, который можно запускать на любом компьютере (рис.13).

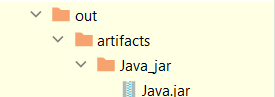


Рисунок 13 – Сгенерированный jar-файл

# **РЕЗУЛЬТАТЫ РАБОТЫ**

В основном окне во вкладке «Студенты» доступны данные для ввода такие как ФИО, пол, курс, номер группы и мобильный телефон студента (рис. 14).

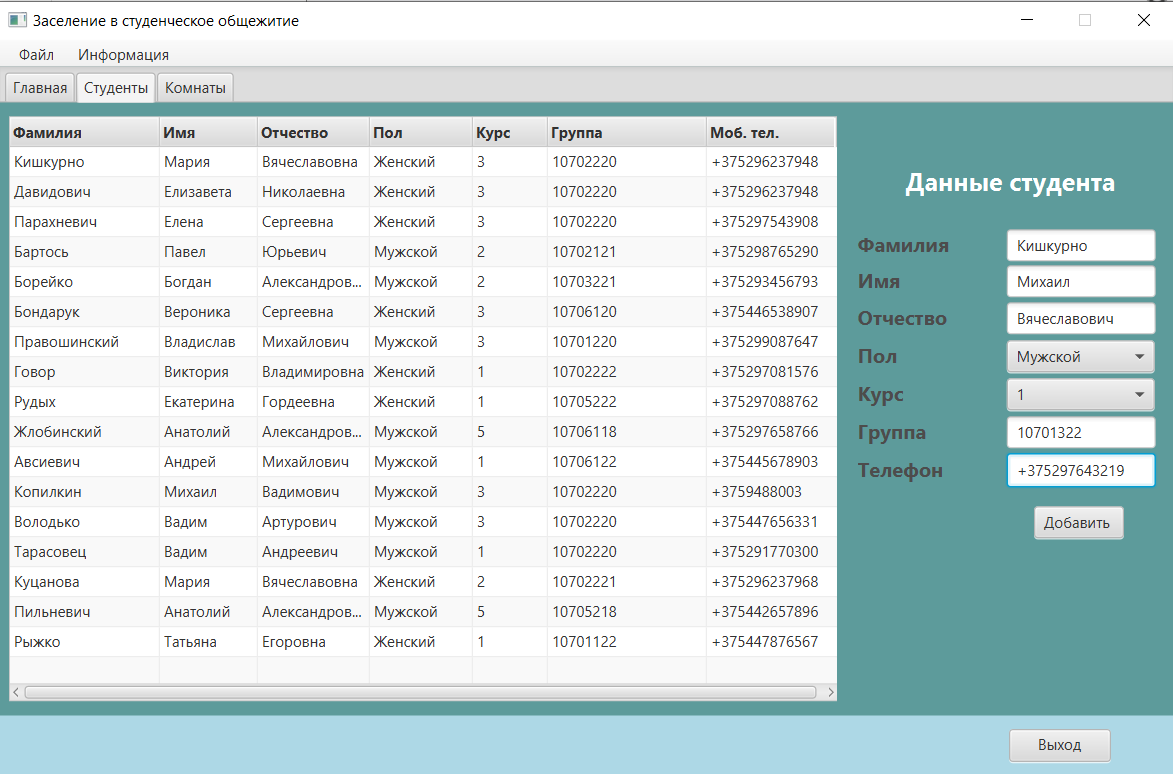


Рисунок 14 – Ввод данных студента

После нажатия на кнопку «Добавить», программа записывает и сохраняет введённые данные о студенте в базу данных, а также обновляет список на экране (рис.15).

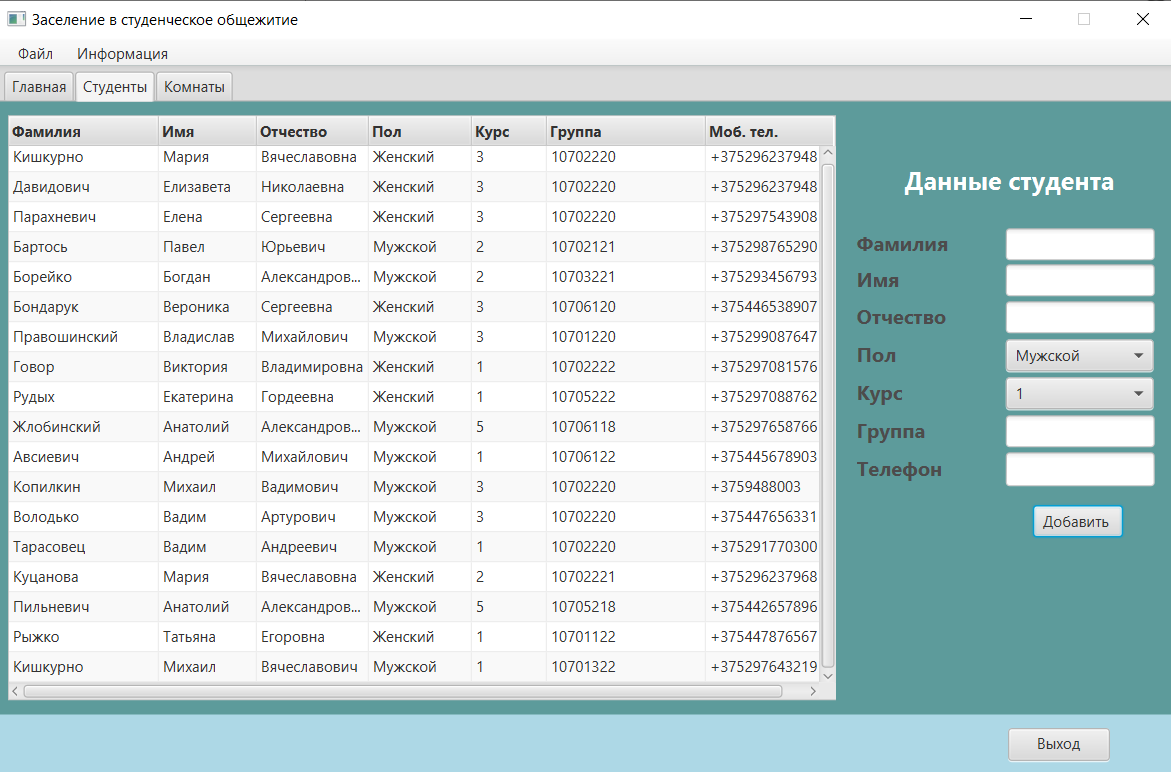


Рисунок 15 – Добавление студента в список

При попытке второй раз добавить ранее добавленного студента появится сообщение об ошибке (рис.16)

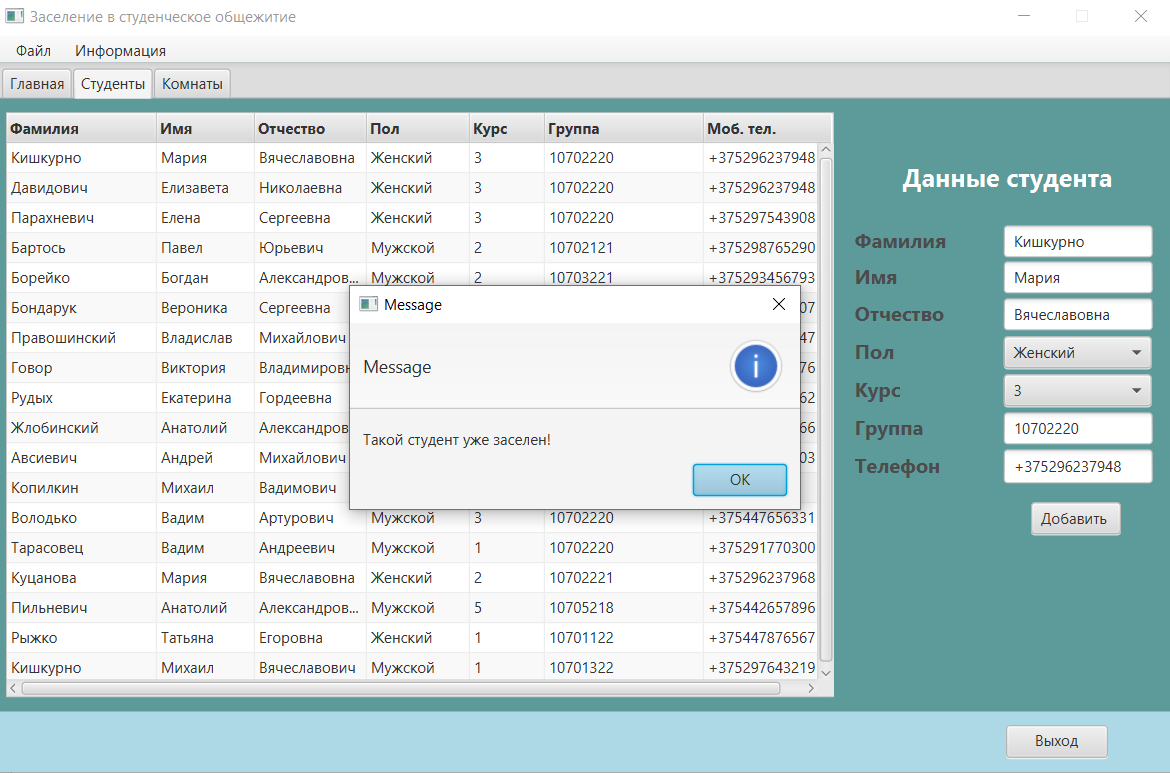


Рисунок 16 – Добавление ранее добавленного студента

При попытке добавить студента, не введя все его данные, появится сообщение об ошибке (рис.17)

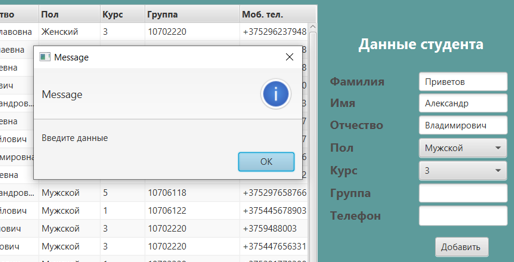


Рисунок 17– Попытка неполного ввода данных

Во складке «Комнаты» изначально можно видеть информацию о ранее заселённых в комнаты студентов, о заполнении, перемещаясь по этажам. Чтобы увидеть актуальную информацию о заполнении комнат из вкладки «Комнаты» нужно нажать кнопку «Обновить» (рис.18).



Рисунок 18 – Заполнение комнат на 4 этаже

Пользователь может просмотреть детальную информацию о студентах, заселённых в комнату, нажав по ячейке с соответствующим номером, при этом появится окно комнаты (рис.19)

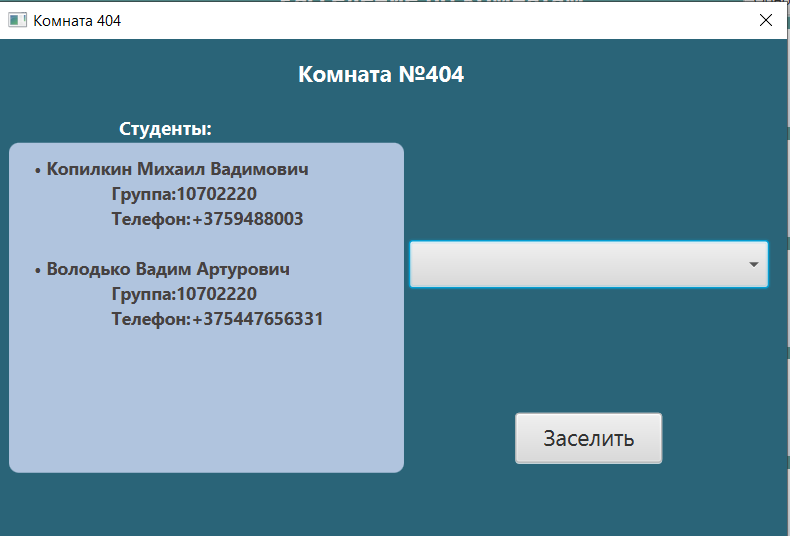


Рисунок 19 – Окно 404 комнаты

Пользователь может заселить в комнату студента, который ранее не был распределён в другую комнату. Выбрать студента для заселения можно в окне интересующей комнаты и добавить в список её проживающих, нажав на кнопку «Заселить» (рис.20-22)

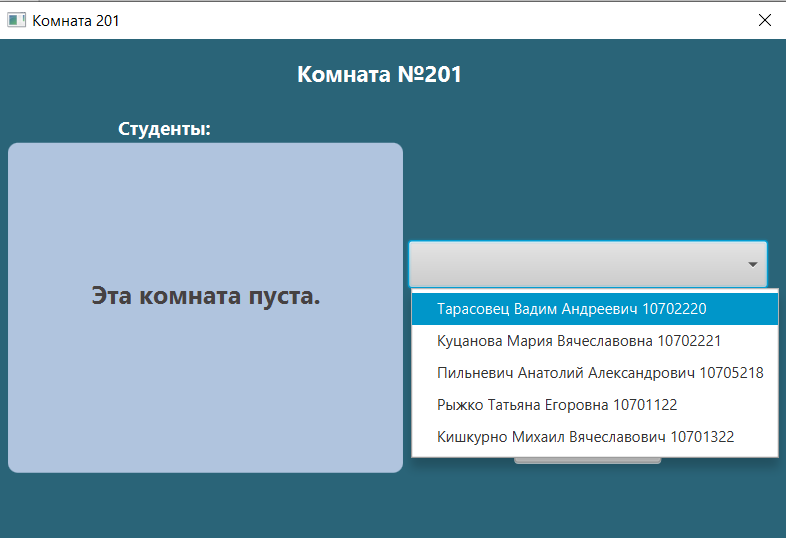


Рисунок 20 – Выбор студента для заселения

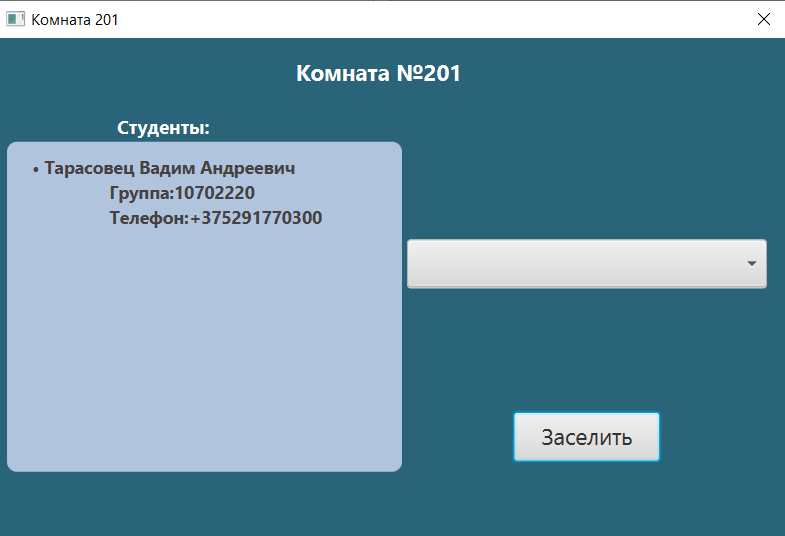


Рисунок 21 – Заселение студента в комнату №201



Рисунок 22 – Обновленная информация о состоянии комнат

При первом заселении в комнату ей присваивается определённый гендер: мужской или женский, то в мужской комнате для подселения будет предложен список студентов мужского пола, а в женской, соответственно список студентов женского пола (рис. 23-24)

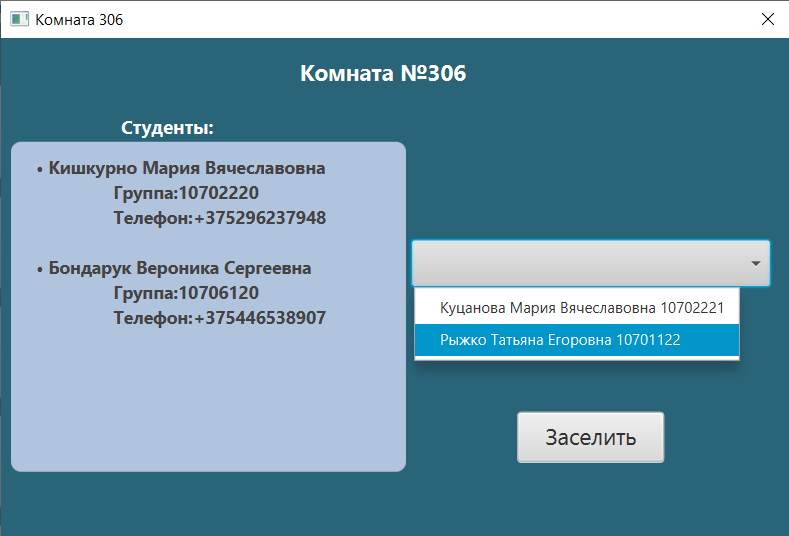


Рисунок 23 – Женская комната

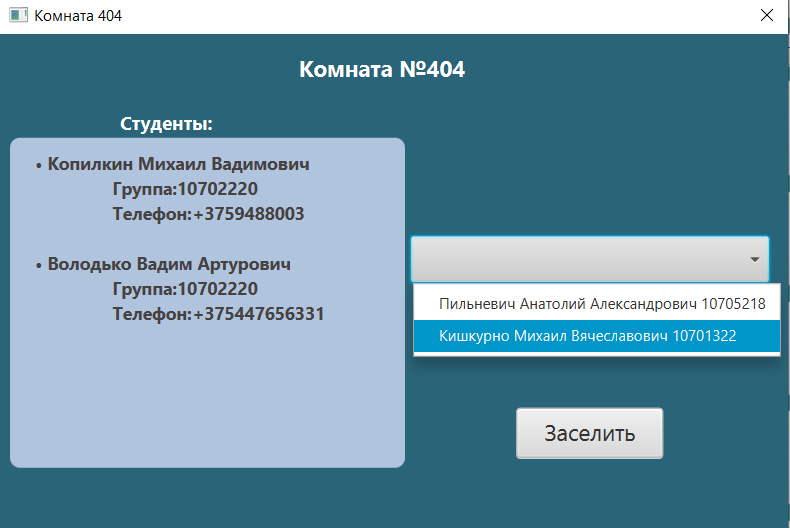


Рисунок 24 – Мужская комната

В одной комнате может проживать не более 3 студентов. Если комната полностью укомплектована, функция заселения нового студента будет заблокирована (рис.25)

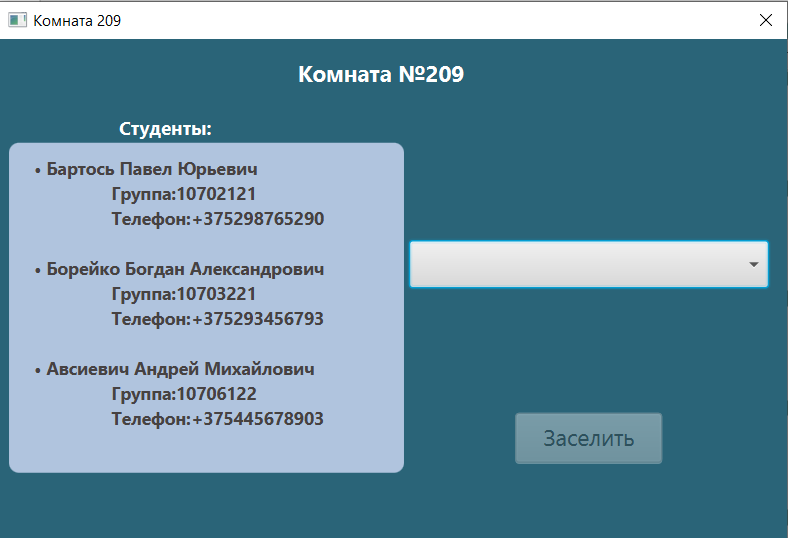


Рисунок 25 – Полностью заселённая комната

Вся введённая информация: о студентах и о расселении по комнатам сохраняется в базе данных.

Выход из программы может осуществляться по кнопке «Выход» в стартовом и основном окне, используя меню «Файл – Выход», либо спустя 60 секунд бездействия.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы было разработано оконное приложение «Заселение студентов в общежитие» на языке программирования Java. Графический интерфейс разработан с использованием платформы JavaFX. Использована СУБД MySQL.

Реализована возможность ввода данных студента, расселение студентов по комнатам, учитывая пол студентов и вместимость комнаты. Также имеется возможность предосмотра информации о заполнении комнат на выбранном этаже и получение более детальной информации о проживающих в определённой комнате студентах. Реализовано сохранение и извлечение данных из базы данных.

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Кей С.Хорстман – Java. Библиотека профессионала.
2. <https://drive.google.com/drive/shared-with-me> - Методические указания к оформлению курсовой работы, В.В. Сидорик.
3. Курс по JavaFX.[Электронный ресурс] – Режим доступа: <https://proglib.io/p/21-urok-po-javafx-dlya-nachinayushchih-2019-12-06>, свободный.
4. <https://javadevblog.com/sozdaem-yspolnyaemyj-jar-v-intellij-idea.html> - Cоздание jar-файлов.
5. Документация языка программирования Java [Электронный ресурс] – Режим доступа: https://docs.oracle.com/en/java/, свободный.

# **ПРИЛОЖЕНИЕ 1**

**Main.java**

package sample;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
*/\*\*  
 \* Класс Main, запускающий приложение  
 \*  
 \* @author Kishkurno M.V.  
 \* @version 1.1  
 \*/*public class Main extends Application {  
  
 */\*\*  
 \* Метод запускающий приложение  
 \* Принимает путь к стартовому окну  
 \* и размер окна  
 \* @param primaryStage - начальный Stage  
 \* @throws Exception - обработка исключений при создании  
 \*/* @Override  
 public void start(Stage primaryStage) throws Exception{  
 Parent root = FXMLLoader.*load*(getClass().getResource("../resources/sample.fxml"));  
 primaryStage.setTitle("Заселение в студенческое общежитие");  
 primaryStage.setScene(new Scene(root, 700, 450));  
 primaryStage.setResizable(false);  
 primaryStage.show();  
 }  
  
 */\*\*  
 \* Метод, являющийся входной точкой в приложение  
 \* В нём вызывается метод launch(args)  
 \* для передачи аргументов  
 \* @param args   
 \*/* public static void main(String[] args) {  
 *launch*(args);  
 }  
}

**AboutAuthor.java**

package sample;

import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.stage.Stage;  
  
*/\*\*  
 \* Класс для окна Об авторе  
 \* Содержит кнопку возврата "Назад"  
 \*  
 \* @author Kishkurno M.V.  
 \* @version 1.1  
 \*/*public class AboutAuthor {  
  
 @FXML  
 private Button btnBackAuthor;  
  
 */\*\*  
 \* Обработка события, происходящего при  
 \* нажатии кнопки кнопки Назад  
 \* @param event - событие  
 \*/* @FXML  
 void handleCloseButtonAction(ActionEvent event) {  
 Stage stage = (Stage) btnBackAuthor.getScene().getWindow();  
 stage.close();  
 }  
}

**AboutProgram.java**

package sample;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.stage.Stage;  
  
*/\*\*  
 \* Класс для окна О программе  
 \* Cодержит кнопку возврата "Назад"  
 \*  
 \* @author Kishkurno M.V.  
 \* @version 1.1  
 \*/*public class AboutProgram {  
 @FXML  
 private Button btnBackProgram;  
  
 */\*\*  
 \* Обработка события, происходящего при  
 \* нажатии кнопки кнопки "Назад"  
 \* @param event - событие  
 \*/* @FXML  
 void handleCloseButtonAction(ActionEvent event) {  
 Stage stage = (Stage) btnBackProgram.getScene().getWindow();  
 stage.close();  
 }  
}

**MainWindow.java**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.layout.GridPane;  
import javafx.scene.layout.Pane;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import java.io.IOException;  
  
*/\*\*  
 \* Класс главного окна, содержащий весь его функционал  
 \*  
 \* @author Kishkurno M.V.  
 \* @version 1.1  
 \*/*public class MainWindow {  
 */\*\*  
 \* Метод, с помощью которого открываются  
 \* информационные окна Об авторе и О программе  
 \*  
 \* @param windowFormName - название окна  
 \* @param windowFormTitle - заголовок окна  
 \* @throws IOException - обрабытывает исключения при вводе/выводе данных  
 \*/* private void openInformationWindow(String windowFormName, String windowFormTitle)  
 throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("../resources/" + windowFormName + ".fxml"));  
 Parent parent = fxmlLoader.load();  
 Scene scene = new Scene(parent);  
 Stage stage = new Stage();  
 stage.initModality(Modality.*APPLICATION\_MODAL*);  
 stage.setScene(scene);  
 stage.setTitle(windowFormTitle);  
 stage.setResizable(false);  
 stage.showAndWait();  
 }  
  
 private Alert informationAlert = new Alert(Alert.AlertType.*INFORMATION*);  
  
 ObservableList<Student> studentsList = FXCollections.*observableArrayList*(); ;  
 ObservableList<Room> roomsList = FXCollections.*observableArrayList*();  
  
 */\*\*  
 \* Метод, обрабатывающий введенные пользователем данные  
 \* о заселяемом студенте  
 \*/* private void addStudent(){  
 String firstName = txtFirstName.getText().trim();  
 String secondName = txtSecondName.getText().trim();  
 String patronymic = txtPatronymic.getText().trim();  
 String gender = cmbGender.getValue();  
 int course = cmbCourse.getValue();  
 String group = txtGroup.getText().trim();  
 String phone = txtPhone.getText();  
  
 if(firstName.equals("") || secondName.equals("") || patronymic.equals("") || group.equals("")|| phone.equals("")){  
 informationAlert.setContentText("Введите данные");  
 informationAlert.showAndWait();  
 }  
 else {  
 Boolean isContains = false;  
 for(int i = 0; i < studentsList.size(); i++){  
 if(studentsList.get(i).getFirstName().equals(firstName) && studentsList.get(i).getSecondName().equals(secondName)  
 && studentsList.get(i).getPatronymic().equals(patronymic)){  
 informationAlert.setContentText("Такой студент уже заселен!");  
 informationAlert.showAndWait();  
 isContains = true;  
 break;  
 }  
 }  
 if(!isContains){  
 DbHandler.*addStudentToDb*(secondName,firstName,patronymic,gender,course,group,phone);  
 Student newStudent = new Student(secondName, firstName, patronymic, gender, course, group,phone);  
 studentsList.add(newStudent);  
 }  
 }  
 txtFirstName.setText("");  
 txtSecondName.setText("");  
 txtPatronymic.setText("");  
 txtGroup.setText("");  
 txtPhone.setText("");  
 }  
  
 */\*\*  
 \* Метод, завершающий и закрывающий окна программы  
 \* @param event - событие  
 \*/* private void closeStage(ActionEvent event) {  
 Node source = (Node)event.getSource();  
 Stage stage = (Stage)source.getScene().getWindow();  
 stage.close();  
 }  
  
 @FXML  
 private MenuItem mniAboutAuthorInfo;  
  
 @FXML  
 private MenuItem mniAboutProgramInfo;  
  
 @FXML  
 private Button btnAddStudent;  
  
 @FXML  
 private ChoiceBox<Integer> cmbCourse;  
  
 @FXML  
 private TableColumn<Student, Integer> courseStudentColumn;  
  
 @FXML  
 private TableColumn<Student, String> firstNameStudentColumn;  
  
 @FXML  
 private TextField txtFirstName;  
  
 @FXML  
 private TextField txtSecondName;  
  
 @FXML  
 private TextField txtPatronymic;  
  
 @FXML  
 private TextField txtGroup;  
  
 @FXML  
 private TextField txtPhone;  
  
 @FXML  
 private ChoiceBox<String > cmbGender;  
  
 @FXML  
 private TableColumn<Student, String> genderStudentColumn;  
  
 @FXML  
 private TableColumn<Student, Integer> groupStudentColumn;  
 @FXML  
 private Button btnMainExit;  
  
 @FXML  
 private MenuItem mniExitFile;  
 @FXML  
 private TableColumn<Student, String> patronymicStudentColumn;  
  
 @FXML  
 private TableColumn<Student, String> secondNameStudentColumn;  
  
 @FXML  
 private TableColumn<Student, String> phoneStudentColumn;  
  
 @FXML  
 private ChoiceBox<Integer> cmbFloor;  
  
 @FXML  
 private TableView<Student> studentTableView;  
  
 @FXML  
 private GridPane gridRooms;  
  
 @FXML  
 private Button btnUpdateRooms;  
  
 */\*\*  
 \* Метод, отвечающий за инициализацию полей  
 \*/* @FXML  
 void initialize() {  
  
 *//Инициализация поля Пол* ObservableList<String> gender = FXCollections.*observableArrayList*("Женский","Мужской");  
 cmbGender.setItems(gender);  
 cmbGender.setValue(gender.get(0));  
  
 *//Инициализация поля Курс* ObservableList<Integer> courses = FXCollections.*observableArrayList*(1, 2, 3, 4, 5);  
 cmbCourse.setItems(courses);  
 cmbCourse.setValue(courses.get(0));  
  
 *//Инициализация поля Этаж* ObservableList<Integer> floors = FXCollections.*observableArrayList*(1, 2, 3, 4, 5);  
 cmbFloor.setItems(floors);  
 cmbFloor.setValue(floors.get(0));  
  
 */\* Передача параметров в метод openInformationWindow  
 для открытия окна Об авторе  
 \*/* mniAboutAuthorInfo.setOnAction(event -> {  
 try {  
 openInformationWindow("aboutAuthor", "Об авторе");  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 });  
  
 */\* Передача параметров в метод openInformationWindow  
 для открытия окна О программе  
 \*/* mniAboutProgramInfo.setOnAction(event -> {  
 try {  
 openInformationWindow("aboutProgram", "О программе");  
 } catch (IOException e) {  
 e.printStackTrace();  
 } });  
  
 *//вызов метода добавления студента при нажатии на кнопку Добавить* btnAddStudent.setOnAction(event -> {  
 addStudent();  
 studentsList = DbHandler.*getStudentsFromDB*();  
 });  
  
 studentsList = DbHandler.*getStudentsFromDB*();  
  
 cmbFloor.setOnAction(event -> {  
 renderRooms();  
 });  
  
 btnUpdateRooms.setOnAction(event -> {  
 renderRooms();  
 });  
  
 *//кнопка выхода* btnMainExit.setOnAction(event -> {  
 closeStage(event);  
 });  
  
 mniExitFile.setOnAction((ActionEvent t) -> {  
 System.*exit*(0);  
 });  
  
 renderRooms();  
  
 firstNameStudentColumn.setCellValueFactory(new PropertyValueFactory<Student, String>("firstName"));  
 secondNameStudentColumn.setCellValueFactory(new PropertyValueFactory<Student, String>("secondName"));  
 patronymicStudentColumn.setCellValueFactory(new PropertyValueFactory<Student, String>("patronymic"));  
 genderStudentColumn.setCellValueFactory(new PropertyValueFactory<Student, String>("gender"));  
 courseStudentColumn.setCellValueFactory(new PropertyValueFactory<Student, Integer>("course"));  
 groupStudentColumn.setCellValueFactory(new PropertyValueFactory<Student, Integer>("group"));  
 phoneStudentColumn.setCellValueFactory(new PropertyValueFactory<Student, String>("phone"));  
  
 studentTableView.setItems(studentsList);  
 }  
  
 */\*\*  
 \* Метод, отвечающий за размещение комнат на окне  
 \* и за обращение к ним  
 \*/* public void renderRooms(){  
 gridRooms.getChildren().clear();  
 roomsList = DbHandler.*getRoomsFromDB*(cmbFloor.getValue());  
 int row = 0;  
 int colum = 0;  
 for (Room room : roomsList){  
 if (row > 4){  
 colum =1;  
 row=0;  
 }  
 Label lblRoomNumber = new Label(Integer.*toString*(room.getRoomNumber()));  
 lblRoomNumber.setLayoutX(8);  
 lblRoomNumber.setLayoutY(1);  
 Label labelStudentNames = new Label("");  
 labelStudentNames.setLayoutX(90);  
 labelStudentNames.setLayoutY(14);  
  
 for (Student student: room.getStudents()){  
 labelStudentNames.setText(labelStudentNames.getText()+ student.getFirstName()+" "+ student.getSecondName().charAt(0) +"."+ student.getPatronymic().charAt(0) +"\n");  
 }  
  
 Pane paneRoom = new Pane(lblRoomNumber,labelStudentNames);  
 paneRoom.getStyleClass().add("room");  
  
 paneRoom.setOnMouseClicked(mouseEvent ->{  
 try {  
 RoomState.*roomNumber* = Integer.*toString*(room.getRoomNumber());  
 RoomState.*roomGender*= room.getRoomGender();  
 RoomState.*studentsCount* = room.getStudentsCount();  
  
 openInformationWindow("roomInfo", "Комната "+ Integer.*toString*(room.getRoomNumber()) );  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 });  
 gridRooms.add(paneRoom,colum,row);  
 row++;  
 }  
 }  
}

**Student.java**

package sample;  
  
*/\*\*  
 \* Класс, описывающий заселяемого студента  
 \*  
 \* @author Kishkurno Maria  
 \* @version 1.1  
 \*/*public class Student {  
 private String firstName;  
 private String secondName;  
 private String patronymic;  
 private String gender;  
 private int course;  
 private String group;  
 private String phone;  
 private int roomNumber;  
 private boolean isSettled;  
  
 */\*\*  
 \* Конструктор для создания экземпляра студента  
 \*  
 \* @param secondName - Фамилия  
 \* @param firstName - Имя  
 \* @param patronymic - Отчество  
 \* @param gender - пол  
 \* @param course - курс  
 \* @param group - группа  
 \* @param phone - номер телефона  
 \*/* public Student(String secondName, String firstName, String patronymic, String gender, int course, String group, String phone) {  
 this.firstName = firstName;  
 this.secondName = secondName;  
 this.patronymic = patronymic;  
 this.gender = gender;  
 this.course = course;  
 this.group = group;  
 this.phone = phone;  
 isSettled = false;  
 }  
  
 public void setIsSettled(boolean isSettled) {  
 this.isSettled = isSettled;  
 }  
  
 */\*\*  
 \* Метод присваивающий полю комнаты номер  
 \* @param roomNumber - номер комнаты  
 \*/* public void setRoomNumber(int roomNumber){  
 this.roomNumber = roomNumber;  
 }  
  
 */\*\*  
 \* Метод, позволяющий получать имя студента  
 \* @return student's name  
 \*/* public String getFirstName() {  
 return firstName;  
 }  
  
 */\*\*  
 \* Метод. позволяющий получать фамилию студента  
 \* @return secondname  
 \*/* public String getSecondName() {  
 return secondName;  
 }  
  
 */\*\*  
 \* Метод, позволяющий получать отчество студента  
 \* @return patronymic  
 \*/* public String getPatronymic() {  
 return patronymic;  
 }  
  
 */\*\*  
 \* Метод, позволяющий получать пол студента  
 \* @return gender  
 \*/* public String getGender() {  
 return gender;  
 }  
  
 */\*\*  
 \* Метод, позволяющий получать курс студента  
 \* @return course  
 \*/* public int getCourse() {  
 return course;  
 }  
  
 */\*\*  
 \* Метод, позволяющий получить номер группы студента  
 \* @return group  
 \*/* public String getGroup() {  
 return group;  
 }  
  
 */\*\*  
 \* Метод, позволяющий получить номер телефона студента  
 \* @return phone  
 \*/* public String getPhone() {  
 return phone;  
 }  
  
 */\*\*  
 \* Метод, возвращающий состояние студента  
 \* заселён/ не заселён  
 \* @return isSettled  
 \*/* public boolean getIsSettled() {  
 return isSettled;  
 }  
}

**Room.java**

package sample;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Класс распределения студентов по комнатам,  
 \* содержащий основные данные  
 \*  
 \* @author Kishkurno M.V.  
 \* @version 1.1  
 \*/*public class Room {  
 private List<Student> students = new ArrayList<>();  
 private int roomNumber;  
 private int floor;  
 private String roomGender;  
 private int studentsCount = 0;  
  
  
 */\*\*  
 \* Конструктор для создания экземпляра  
 \* заселённой комнаты  
 \*  
 \* @param floor - этаж  
 \* @param roomNumber - номер комнаты  
 \*/* public Room(int floor, int roomNumber) {  
 this.floor = floor;  
 this.roomNumber = roomNumber;  
 studentsCount++;  
 }  
  
 */\*\*  
 \* Метод, присваивающий этажу номер  
 \* @param floor - этаж  
 \*/* public void setFloor(int floor) {  
 this.floor = floor;  
 }  
  
 */\*\*  
 \* Метод, присваивающий комнате номер  
 \* @param roomNumber - номер комнаты  
 \*/* public void setRoomNumber(int roomNumber) {  
 this.roomNumber = roomNumber;  
 }  
  
 */\*\*  
 \* Метод, отвечающий за заполнение списка студентов  
 \* заселённых в одну комнату  
 \* @param student - студент  
 \*/* public void setStudent(Student student) {  
 students.add(student);  
 }  
  
 */\*\*  
 \* Метод, присваивающий всей комнате гендер  
 \* @param roomGender - гендер комнаты  
 \*/* public void setRoomGender(String roomGender){  
 this.roomGender = roomGender;  
 }  
  
 */\*\*  
 \* Метод, который принимает количесвто студентов  
 \* @param studentsCount - количество студентов  
 \*/* public void setStudentsCount(int studentsCount){  
 this.studentsCount = studentsCount;  
 }  
  
 */\*\*  
 \* Метод, возвращающий список студентов  
 \* проживающей в одной комнате  
 \* @return list of students  
 \*/* public List<Student> getStudents() {  
 return students;  
 }  
  
 */\*\*  
 \* Метод, возвращающий номер этажа  
 \* @return number of floor  
 \*/* public int getFloor() {  
 return floor;  
 }  
  
 */\*\*  
 \* Метод, возвращающий номер комнаты  
 \* @return number of room  
 \*/* public int getRoomNumber() {  
 return roomNumber;  
 }  
  
 */\*\*  
 \* Метод, возвращающий гендер комнаты  
 \* @return rooms gender  
 \*/* public String getRoomGender(){  
 return roomGender;  
 }  
  
 */\*\*  
 \* Метод, возвращающий количество студентов  
 \* @return count of students  
 \*/* public int getStudentsCount(){  
 return studentsCount;  
 }  
}

**RoomInfo.java**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ChoiceBox;  
import javafx.scene.control.Label;  
import javafx.scene.text.Font;  
import javafx.scene.text.FontPosture;  
import javafx.scene.text.FontWeight;  
import java.sql.\*;  
  
*/\*\*  
 \* Класс, описывающий выбранную комнату  
 \*  
 \* @author Kishkurno M.V.  
 \* @version 1.1  
 \*/*public class RoomInfo {  
 Connection connection ;  
 ResultSet resultSet ;  
 Statement statement ;  
 ObservableList<Student> studentsInRoomList = FXCollections.*observableArrayList*();  
 ObservableList<Student> studentsList = FXCollections.*observableArrayList*();  
  
 */\*\*  
 \* Метод, позволяющий получать информацию  
 \* об имеющихся студентах  
 \*/* private void getStudentsFromDB(){  
 try {  
 connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 statement = connection.createStatement();  
 if(RoomState.*roomGender*.isEmpty()){  
 resultSet =statement.executeQuery("select \* from hostel.students where isSettled=0");  
 }  
 else {  
 resultSet =statement.executeQuery("select \* from hostel.students where isSettled=0 and gender=\""+RoomState.*roomGender*+"\"");  
 }  
  
 studentsList.clear();  
  
 while (resultSet.next())  
 {  
 Student newStudent = new Student( resultSet.getString("secondName"),  
 resultSet.getString("firstName"),  
 resultSet.getString("patronymic"),  
 resultSet.getString("gender"),  
 resultSet.getInt("course"),  
 resultSet.getString("group"),  
 resultSet.getString("phone"));  
 studentsList.add(newStudent);  
 }  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 Label lblRoomNumber;  
  
 @FXML  
 Label lblStudents;  
  
 @FXML  
 Button btnAddStudentToRoom;  
  
 @FXML  
 ChoiceBox cmbStudent;  
  
 */\*\*  
 \* Метод, отвечающий за инициализацию  
 \* информации о студентах и комнатах  
 \*/* @FXML  
 void initialize() {  
 lblRoomNumber.setText(RoomState.*roomNumber*);  
 studentsInRoomList = DbHandler.*getStudentsByRoomNumberFromDB*();  
 renderStudents();  
 getStudentsFromDB();  
  
 for (Student student:studentsList){  
 cmbStudent.getItems().add(student.getFirstName()+" " +student.getSecondName()+" " +student.getPatronymic() +" " +student.getGroup());  
 }  
  
 btnAddStudentToRoom.setOnAction(event ->{  
 String studentFromChoiceBox = cmbStudent.getValue().toString();  
 String[] splited =studentFromChoiceBox.split("\\s+");  
 Student newStudentInRoom = null;  
 for(Student student:studentsList){  
 if(student.getFirstName().equals(splited[0]) && student.getSecondName().equals(splited[1])  
 && student.getPatronymic().equals(splited[2]) && student.getGroup().equals(splited[3])){  
 newStudentInRoom = student;  
 }  
 }  
try {  
 final String SQL\_UPDATE\_ROOM = "UPDATE students SET secondName=?, firstName=?, patronymic=?, gender=?, course=?, students.group=?, phone=?, roomNumber=?, isSettled=? " +  
 "WHERE secondName=? AND firstName=? AND patronymic=? AND students.group=?";  
 connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 PreparedStatement statement = connection.prepareStatement(SQL\_UPDATE\_ROOM, Statement.*RETURN\_GENERATED\_KEYS*);  
 statement.setString(1, newStudentInRoom.getSecondName());  
 statement.setString(2, newStudentInRoom.getFirstName());  
 statement.setString(3, newStudentInRoom.getPatronymic());  
 statement.setString(4, newStudentInRoom.getGender());  
 statement.setInt(5,newStudentInRoom.getCourse());  
 statement.setString(6, newStudentInRoom.getGroup());  
 statement.setString(7, newStudentInRoom.getPhone());  
 statement.setInt(8, Integer.*parseInt*(RoomState.*roomNumber*));  
 statement.setBoolean(9, true);  
 statement.setString(10, newStudentInRoom.getSecondName());  
 statement.setString(11, newStudentInRoom.getFirstName());  
 statement.setString(12, newStudentInRoom.getPatronymic());  
 statement.setString(13, newStudentInRoom.getGroup());  
  
 int rowsUpdate = statement.executeUpdate();  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
  
 RoomState.*studentsCount*++;  
try {  
 final String SQL\_INSERT\_ROOM = "UPDATE rooms SET roomGender=?, studentsCount=? " +  
 "WHERE roomNumber=?";  
 connection = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 PreparedStatement statement = connection.prepareStatement(SQL\_INSERT\_ROOM, Statement.*RETURN\_GENERATED\_KEYS*);  
 statement.setString(1, newStudentInRoom.getGender());  
 statement.setInt(2, RoomState.*studentsCount*);  
 statement.setInt(3, Integer.*parseInt*(RoomState.*roomNumber*));  
 int rowsUpdate = statement.executeUpdate();  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
  
 RoomState.*roomGender* = newStudentInRoom.getGender();  
 getStudentsFromDB();  
 studentsInRoomList = DbHandler.*getStudentsByRoomNumberFromDB*();  
 renderStudents();  
 cmbStudent.getItems().clear();  
  
 for (Student student:studentsList){  
 cmbStudent.getItems().add(student.getFirstName()+" " +student.getSecondName()+" " +student.getPatronymic() +" " +student.getGroup());  
 }  
 });  
 }  
  
 */\*\*  
 \* Метод, выводящий список студентов  
 \* в определённой комнате  
 \* и их личную информацию  
 \*/* private void renderStudents(){  
  
 if(RoomState.*studentsCount* ==3){  
 btnAddStudentToRoom.setDisable(true);  
 }  
  
 if(studentsInRoomList.isEmpty()){  
 lblStudents.setFont(Font.*font*("System", FontWeight.*BOLD*, FontPosture.*REGULAR*,20));  
 lblStudents.setLayoutX(75);  
 lblStudents.setLayoutY(190);  
 lblStudents.setText("Эта комната пуста.");  
 }  
 else {  
 lblStudents.setLayoutX(28);  
 lblStudents.setLayoutY(83);  
 lblStudents.setText("");  
 lblStudents.setFont(Font.*font*("System", FontWeight.*BOLD*, FontPosture.*REGULAR*,14));  
 for (Student student: studentsInRoomList){  
 lblStudents.setText(lblStudents.getText()+"• "+student.getFirstName() +" " + student.getSecondName() +" " + student.getPatronymic()+ "\n\t\tГруппа:"+student.getGroup()+ "\n\t\tТелефон:" + student.getPhone()+ "\n\n");  
 }  
 }  
 }  
}

**RoomState.java**

package sample;  
  
*/\*\*  
 \* Класс, отвечающий за базовой состояние каждой комнаты  
 \*  
 \* @author Kishkurno Maria  
 \* @version 1.1  
 \*/*public class RoomState {  
 public static String *roomNumber* ="";  
 public static String *roomGender* ="";  
 public static int *studentsCount* = 0;  
}

**DbHandller.java**

package sample;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import java.sql.\*;  
  
*/\*\*  
 \* Класс, осуществляющий работу  
 \* с базой данных  
 \*  
 \* @author Kishkurno M.V.  
 \* @version 1.1  
 \*/*public class DbHandler {  
 private static Connection *connection* ;  
 private static Connection *extraConnection* ;  
 private static Statement *statement* ;  
 private static Statement *extraStatement* ;  
 private static ResultSet *resultSet* ;  
 private static ResultSet *extraResultSet*;  
  
  
 */\*\*  
 \* Метод, возращающий список всех студентов  
 \* из базы данных  
 \* @return list of students  
 \*/* public static ObservableList<Student> getStudentsFromDB(){  
 ObservableList<Student> studentsList = FXCollections.*observableArrayList*();  
 try {  
 *connection* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 *statement* = *connection*.createStatement();  
 *resultSet* =*statement*.executeQuery("select \* from students");  
 studentsList.clear();  
 while (*resultSet*.next())  
 {  
 Student newStudent = new Student( *resultSet*.getString("secondName"),  
 *resultSet*.getString("firstName"),  
 *resultSet*.getString("patronymic"),  
 *resultSet*.getString("gender"),  
 *resultSet*.getInt("course"),  
 *resultSet*.getString("group"),  
 *resultSet*.getString("phone"));  
  
 studentsList.add(newStudent);  
 }  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
 return studentsList;  
 }  
  
 */\*\*  
 \* Метод, возвращающий список комнат  
 \* на определённом этаже  
 \* с заселёнными в них студентов  
 \* @param floor - этаж  
 \* @return list of rooms  
 \*/* public static ObservableList<Room> getRoomsFromDB(int floor){  
 ObservableList<Room> roomsList = FXCollections.*observableArrayList*();  
 try {  
 *connection* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 *statement* = *connection*.createStatement();  
 *resultSet* =*statement*.executeQuery("select \* from rooms where Floor="+floor);  
  
 roomsList.clear();  
  
 *extraConnection* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 *extraStatement* = *connection*.createStatement();  
 int i =0;  
 while (*resultSet*.next()){  
  
 *extraResultSet* = *extraStatement*.executeQuery("select \* from students where roomNumber="+*resultSet*.getString("roomNumber"));  
 Student student;  
  
 Room room =new Room(*resultSet*.getInt("Floor"),*resultSet*.getInt("roomNumber"));  
 room.setRoomGender(*resultSet*.getString("roomGender"));  
 room.setStudentsCount(*resultSet*.getInt("studentsCount"));  
 roomsList.add(room);  
  
 while (*extraResultSet*.next())  
 {  
 student = new Student(*extraResultSet*.getString("secondName"),  
 *extraResultSet*.getString("firstName"),  
 *extraResultSet*.getString("patronymic"),  
 *extraResultSet*.getString("gender"),  
 *extraResultSet*.getInt("course"),  
 *extraResultSet*.getString("group"),  
 *extraResultSet*.getString("phone"));  
 roomsList.get(i).setStudent(student);  
 }  
 i++;  
 }  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
 return roomsList;  
 }  
  
 */\*\*  
 \* Метод, записывабщий полученную информацию о студенте в базу данных  
 \*  
 \* @param secondName - фамилия  
 \* @param firstName - имя  
 \* @param patronymic - отчество  
 \* @param gender - пол  
 \* @param course - курс  
 \* @param group - группа  
 \* @param phone - номер телефона  
 \*/* public static void addStudentToDb(String secondName,String firstName,String patronymic, String gender, int course,String group,String phone){  
 try {  
 final String SQL\_INSERT\_ROOM = "INSERT INTO students (secondName, firstName, patronymic, gender, course, students.group, phone," +" roomNumber, isSettled) " + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";  
 *connection* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 PreparedStatement statement = *connection*.prepareStatement(SQL\_INSERT\_ROOM, Statement.*RETURN\_GENERATED\_KEYS*);  
 statement.setString(1, secondName);  
 statement.setString(2, firstName);  
 statement.setString(3, patronymic);  
 statement.setString(4, gender);  
 statement.setInt(5, course);  
 statement.setString(6, group);  
 statement.setString(7, phone);  
 statement.setNull(8, Types.*INTEGER* );  
 statement.setBoolean(9, false);  
  
 int rowsUpdate = statement.executeUpdate();  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
 }

*/\*\*  
 \* Метод, возвращающий список студентов и информацию о них  
 \* Заселенных в одну комнату  
 \* @return list of students in room  
 \*/* public static ObservableList<Student> getStudentsByRoomNumberFromDB(){  
 ObservableList<Student> studentsInRoomList = FXCollections.*observableArrayList*();  
 try {  
 *connection* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 *statement* = *connection*.createStatement();  
 *resultSet* =*statement*.executeQuery("select \* from students where roomNumber="+RoomState.*roomNumber*);  
 studentsInRoomList.clear();  
  
 while (*resultSet*.next())  
 {  
 Student newStudent = new Student( *resultSet*.getString("secondName"),  
 *resultSet*.getString("firstName"),  
 *resultSet*.getString("patronymic"),  
 *resultSet*.getString("gender"),  
 *resultSet*.getInt("course"),  
 *resultSet*.getString("group"),  
 *resultSet*.getString("phone"));  
  
 studentsInRoomList.add(newStudent);  
 }  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
 return studentsInRoomList;  
 }  
  
 */\*\*  
 \* Метод, обновляющий информацию о состоянии комнат,  
 \* а именно списке заселенных в комнату студентов  
 \*  
 \* @param newStudentInRoom - новый студент  
 \*/* public static void updateRoom(Student newStudentInRoom){  
 try {  
 final String SQL\_UPDATE\_ROOM = "UPDATE students SET secondName=?, firstName=?, patronymic=?, gender=?, course=?, students.group=?, phone=?, roomNumber=?, isSettled=? " +  
 "WHERE secondName=? AND firstName=? AND patronymic=? AND students.group=?";  
 *connection* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 PreparedStatement statement = *connection*.prepareStatement(SQL\_UPDATE\_ROOM, Statement.*RETURN\_GENERATED\_KEYS*);  
 statement.setString(1, newStudentInRoom.getSecondName());  
 statement.setString(2, newStudentInRoom.getFirstName());  
 statement.setString(3, newStudentInRoom.getPatronymic());  
 statement.setString(4, newStudentInRoom.getGender());  
 statement.setInt(5,newStudentInRoom.getCourse());  
 statement.setString(6, newStudentInRoom.getGroup());  
 statement.setString(7, newStudentInRoom.getPhone());  
 statement.setInt(8, Integer.*parseInt*(RoomState.*roomNumber*));  
 statement.setBoolean(9, true);  
 statement.setString(10, newStudentInRoom.getSecondName());  
 statement.setString(11, newStudentInRoom.getFirstName());  
 statement.setString(12, newStudentInRoom.getPatronymic());  
 statement.setString(13, newStudentInRoom.getGroup());  
  
 int rowsUpdate = statement.executeUpdate();  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Метод, возвращающий список студентов,  
 \* которых можно заселить в выбранную комнату  
 \* @return list of students  
 \*/* public static ObservableList<Student> getStudentsToAddFromDB(){  
 ObservableList<Student> studentsList = FXCollections.*observableArrayList*();  
 try {  
 *connection* = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/hostel", "root", "L1953S");  
 *statement* = *connection*.createStatement();  
 if(RoomState.*roomGender*.isEmpty()){  
 *resultSet* =*statement*.executeQuery("select \* from hostel.students where isSettled=0");  
 }  
 else {  
 *resultSet* =*statement*.executeQuery("select \* from hostel.students where isSettled=0 and gender=\""+RoomState.*roomGender*+"\"");  
 }  
  
 studentsList.clear();  
  
 while (*resultSet*.next())  
 {  
 Student newStudent = new Student( *resultSet*.getString("secondName"),  
 *resultSet*.getString("firstName"),  
 *resultSet*.getString("patronymic"),  
 *resultSet*.getString("gender"),  
 *resultSet*.getInt("course"),  
 *resultSet*.getString("group"),  
 *resultSet*.getString("phone"));  
 studentsList.add(newStudent);  
 }  
 }  
 catch (Exception e)  
 {  
 e.printStackTrace();  
 }  
 return studentsList; }  
}

**Start.java**

package sample;  
  
import javafx.animation.PauseTransition;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.input.InputEvent;  
import javafx.stage.Stage;  
import javafx.util.Duration;  
import java.io.IOException;  
  
*/\*\*  
 \* Класс, отвечающий за работу  
 \* со Stage  
 \*  
 \* @author Kishkurno M.V.  
 \* @version 1.1  
 \*/*public class Start {  
  
 */\*\*  
 \* Метод, закрывающий Stage  
 \* @param event - событие  
 \*/* private void closeStage(ActionEvent event) {  
 Node source = (Node)event.getSource();  
 Stage stage = (Stage)source.getScene().getWindow();  
 stage.close();  
 }  
  
 @FXML  
 private Button btnStartContinue;  
  
 @FXML  
 private Button btnStartExit;  
  
 */\*\* Метод, отвечающий за инициализацию  
 \* параметров Stage  
 \*/* @FXML  
 void initialize(){  
 btnStartContinue.setOnAction(event ->{  
 btnStartContinue.getScene().getWindow().hide();  
  
 FXMLLoader loader = new FXMLLoader();loader.setLocation(getClass().getResource("/resources/mainWindow.fxml"));  
  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setScene(new Scene(root));  
 stage.setTitle("Заселение в студенческое общежитие");  
 *// create transition for logout* Duration delay = Duration.*seconds*(60);  
 PauseTransition transition = new PauseTransition(delay);  
 transition.setOnFinished(evt -> {System.*exit*(0);});  
  
 *// restart transition on user interaction* stage.addEventFilter(InputEvent.*ANY*, evt -> transition.playFromStart());  
 transition.play();  
 stage.setResizable(false);  
 stage.showAndWait();  
 });  
  
 btnStartExit.setOnAction(event -> {  
 closeStage(event);  
 }); }  
}