ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ.ПРОФ.М.А.БОНЧ-БРУЕВИЧА»

(СПбГУТ)

**Кафедра безопасности информационных систем**

**Пояснительная записка к курсовому проекту**

**на тему**

**«»**

по дисциплине «»

Выполнил:

« » г. \_\_\_\_\_\_\_\_\_\_\_/

Принял:

« » г. \_\_\_\_\_\_\_\_\_\_\_/

**Санкт-Петербург  
2020**

Содержание

[Введение 5](#_Toc29893284)

[Цель работы 5](#_Toc29893285)

[Основная часть 5](#_Toc29893286)

[Разработка программного решения 6](#_Toc29893287)

[Заключение 13](#_Toc29893288)

[Список используемых источников. 14](#_Toc29893289)

[Приложение А 15](#_Toc29893290)

[Приложение Б 16](#_Toc29893291)

[Приложение В 22](#_Toc29893292)

# **Введение**

В настоящее время автоматизация бизнес-процессов становится всё более актуальной. Это связано с глобальной компьютеризацией и тем, что компьютеры позволяет решать тривиальные задачи быстро, эффективно, и, главное для бизнеса, дёшево.

В рамках данной курсовой работы была спроектирована и разработана система автоматизации организации работы в школе изучения иностранных языков, а именно процесс управления студентами – добавление новых, удаление существующих, просмотр списка всех студентов.

Актуальность подобной системы обоснована так же тем, что школа может иметь несколько филиалов, и необходим инструмент, позволяющий обрабатывать и хранить данные обо всех филиалах.

# **Цель работы**

Курсовая работа имеет своей целью:

- применение и отработка на практике принципов процедурного и объектно-ориентированного программирования;

- знакомство с современными технологиями разработки;

- разработка конечного кроссплатформенного приложения с графическим интерфейсом.

# **Основная часть**

В результате исследования предметной области было определено, что она описывается в терминах следующих сущностей:

1. Человек. Описывается следующими характеристиками:
   1. Идентификатор
   2. Имя
   3. Фамилия
2. Учебный филиал школы. Представляет из себя здание, в котором проходят занятия. Характеристики:
   1. Идентификатор
   2. Адрес
   3. Номер дома
3. Учебная группа. Представляет из себя группу людей, изучающих определенный язык в определенном филиале школы. Характеристики:
   1. Номер группы
   2. Изучаемый язык
   3. Филиал, в котором проходят занятия
4. Студент. Представляет из себя человека, посещающего занятия в одной из учебных групп.
5. Работник. Является человеком, трудоустроенным в школе, и выполняющим определенные обязанности.

Для посещения занятий студента сначала нужно зарегистрировать. Студент может быть как новым человеком в системе (первый раз записывается на курсы), так и неновым (человек решает записаться на другой курс школы). Студенты школы посещают занятия в группах (группа может состоять из 1-го человека). Каждая группа закреплена за каким-то филиалом школы и её студенты изучают один определенный язык.

В разрабатываемой системе работники являются конечными клиентами, так как вся работа по организации занятий и добавлению, удалению новых студентов выполняется именно работниками школы.

Процесс разработки приложения было решено поделить на 6 этапов:

1. Выбор используемых технологий (База данных и СУБД, ORM-провайдер, платформа для построения GUI);
2. Проектирование архитектуры базы данных;
3. Построение базы данных и проверка корректности выбранной архитектуры;
4. Проектирование архитектуры приложения;
5. Реализация и проверка корректности выбранной архитектуры приложения;
6. Разработка графического интерфейса;
7. Тестирование приложения.

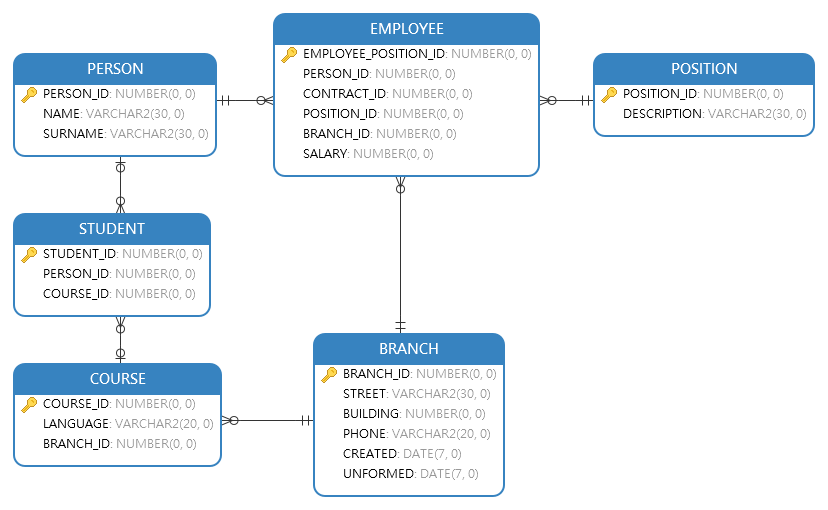
# **Разработка программного решения**

Для реализации продукта было решено использовать следующие технологии:

1. База данных – Oracle;
2. ORM провайдер – Hibernate 5;
3. Платформа для построения GUI – JavaFX;

На этапе начальных этапах было необходимо спроектировать и создать базу данных, позволяющую хранить выявленные на этапе исследования предметной области сущности.

На рисунке 1 представлена ER-модель построенной базы данных. (Data-Definition Language базы данных представлен в приложении А.)



*Рисунок 1. ER-диаграмма спроектированной базы данных.*

После разработки базы данных было необходимо спроектировать архитектуру приложения. Было решено разбить все классы на два пакета.

Первый пакет – Data – представляет из себя классы, отвечающие за взаимодействие с базой данных, а также за их реляционно-объектное отражение. Пакет Data содержит следующие классы:

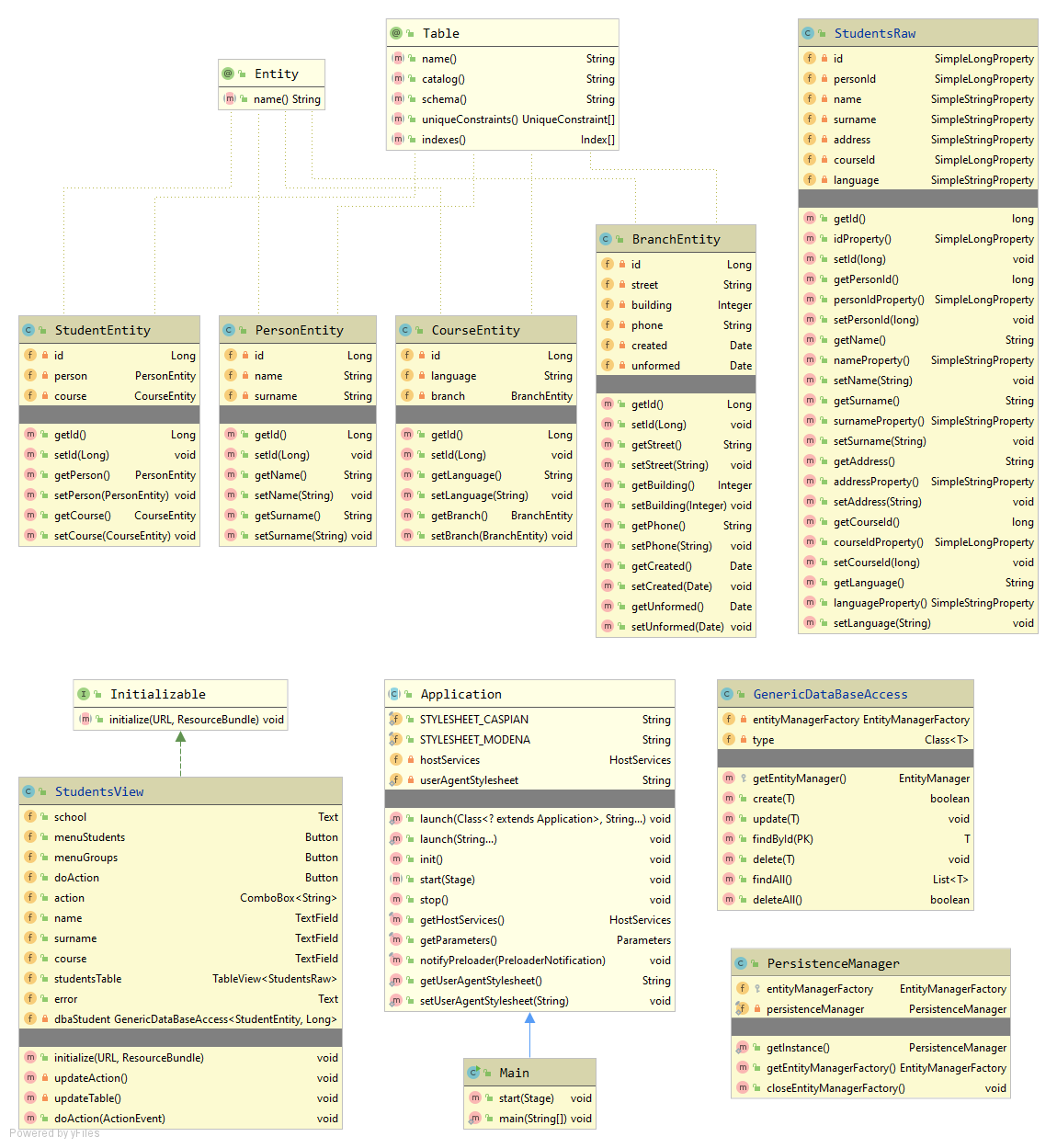
1. PersonEntity – класс, представляющий из себя объектное отображение таблицы “Person” из базы данных и сущность “Человек”.
2. CourseEntity – класс, представляющий из себя объектное отображение таблицы “Course” из базы данных и сущность “Группа”.
3. StudentEntity – класс, представляющий из себя объектное отображение таблицы “Student” из базы данных и сущность “Студент”.
4. BranchEntity – класс, представляющий из себя объектное отображение таблицы “Branch” из базы данных и сущность “Филиал”.
5. GenericDataBaseAccess – класс, позволяющий сгенерировать объект обращения в БД для конкретной сущности.
6. PersistenceManager – класс, необходимый для конфигурации и корректной работы ORM-провайдера.

Второй пакет – View – классы которого отвечают за построение интерактивного графического интерфейса. Пакет View содержит следующие классы:

1. StudentView – класс, отвечающий за корректное отображение данных в GUI приложения.
2. StudentRaw – класс, использующийся для построения сводной таблицы для отображения в GUI на основании информации объектов классов 1-4.

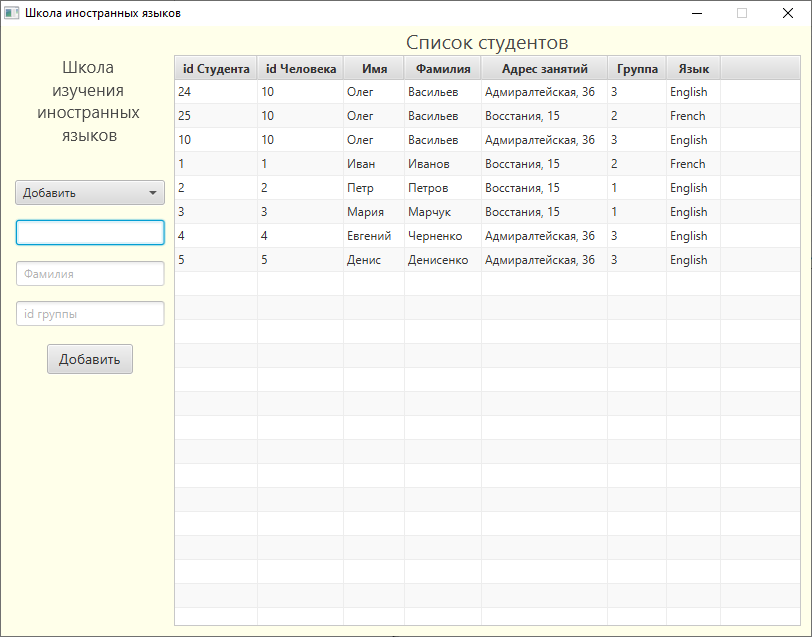
В качестве входной точки приложения использовался класс Main.

На рисунке 2 представлена более подробная UML-диаграмма классов, описывающая все объявленные выше классы, а также их поля и методы (исходный код классов Data.StudentEntity, Data.GenericDataBaseAccess, и View.StudentsView представлены в приложении Б).



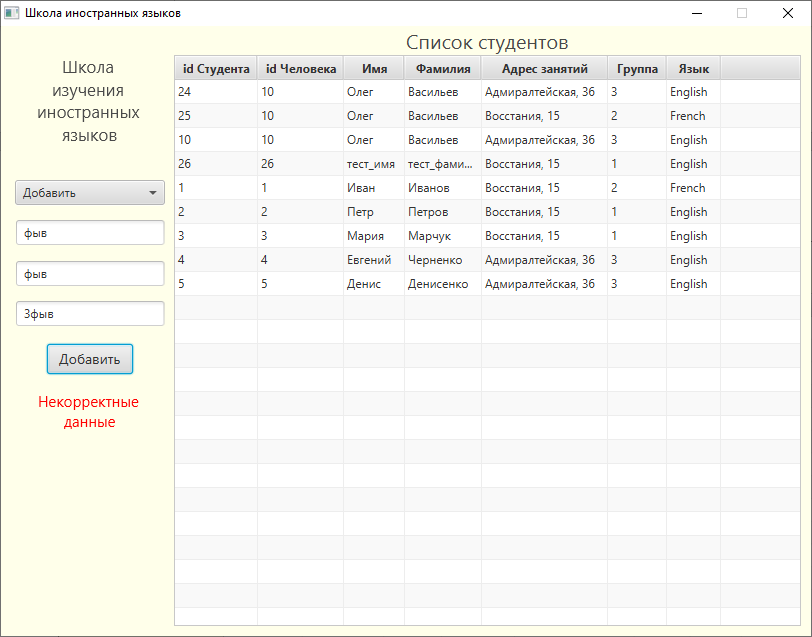
*Рисунок 2. UML-диаграмма классов разработанного приложения.*

На последнем этапе разработки было необходимо разработать дизайн и графический интерфейс для конечного пользователя. Пример разработанного GUI представлен на рисунке 3 (Исходный код конфигурационного fxml файла представлен в приложении В).

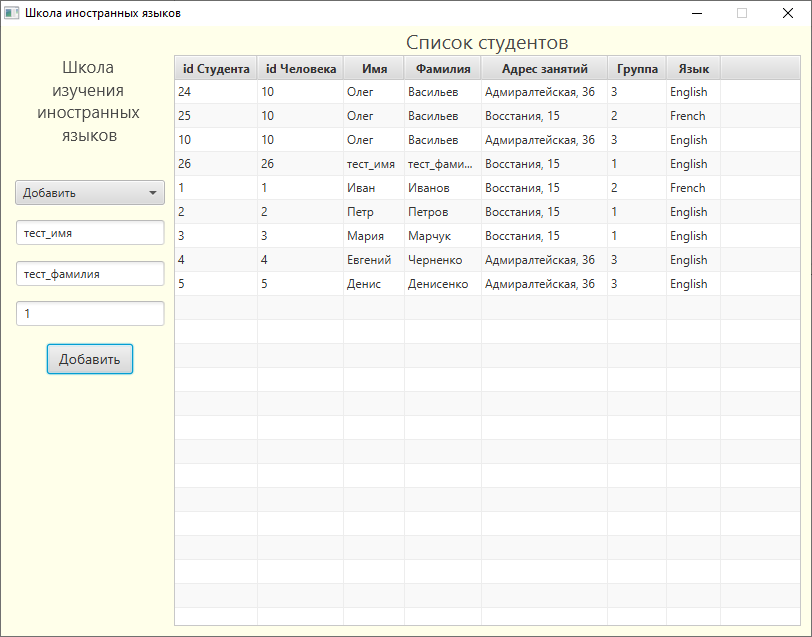


*Рисунок 3. Пример разработанного графического интерфейса.*

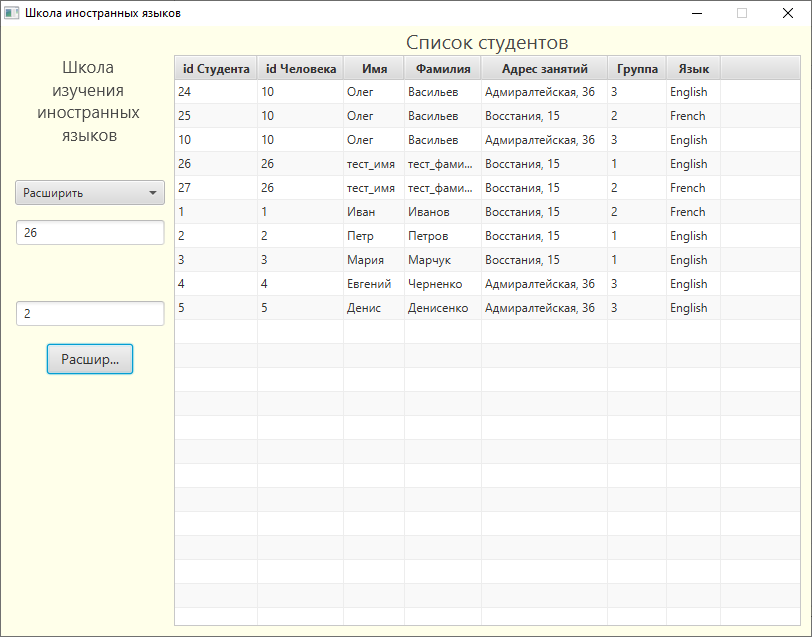
На этапе тестирования приложения был проверен весь функционал приложения и возможность приложения выполнять поставленные перед ним бизнес-задачи. Результаты некоторых тестов представлены на рисунках 4-7.



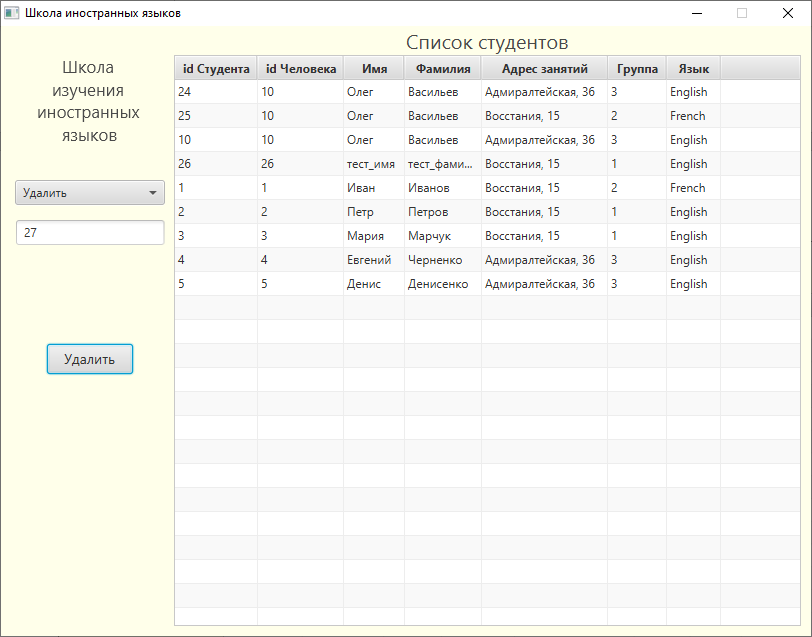
*Рисунок 4. Тест с вводом некорректных значений.*



*Рисунок 5. Тест добавления нового студента.*



*Рисунок 6. Тест с записью существующего человека в новую группу.*



*Рисунок 7. Тест с удалением записи студента.*

Как можно заметить, разработанное приложение корректно выполняет запрограммированную логику.

# **Заключение**

В результате выполнения курсовой работы мной были изучены современные методы разработки кроссплатформенных приложений, спроектировано и разработано приложений с графическим интерфейсом, позволяющее автоматизировать работу школы изучения иностранных языков. На данном этапе разработанное приложение позволяет конечному пользователю – работнику школы – создавать и удалять новых студентов, а также просматривать список всех существующих студентов, используя при этом интуитивно понятный графический интерфейс.

# Список используемых источников.

1. Мишра С., Бьюли А. Секреты Oracle SQL – пер. с англ. – СПб.: Символ-Плюс, 2003 – 386с.
2. Онлайн документация java [Интернет-ресурс] URL: <https://docs.oracle.com/en/java/> (дата обращения: 03.01.2020)
3. Онлайн документация базы данных Oracle [Интернет-ресурс] URL: https://docs.oracle.com/en/database/ (дата обращения: 04.01.2020)
4. Онлайн документация платформы JavaFX [Интернет-ресурс] URL: https://docs.oracle.com/javafx/2/ (дата обращения: 05.01.2020)

# Приложение А

**CREATE TABLE** BRANCH(  
 **BRANCH\_ID NUMBER GENERATED AS IDENTITY PRIMARY KEY**,  
 **STREET VARCHAR**(30) **NOT NULL**,  
 **BUILDING NUMBER NOT NULL**,  
 **PHONE VARCHAR**(20) **NOT NULL**,  
 **CREATED DATE DEFAULT** *CURRENT\_DATE*,  
 **UNFORMED DATE**);  
  
**CREATE TABLE** PERSON(  
 **PERSON\_ID NUMBER GENERATED AS IDENTITY PRIMARY KEY**,  
 **NAME VARCHAR**(30) **NOT NULL**,  
 **SURNAME VARCHAR**(30) **NOT NULL**);  
  
  
**CREATE TABLE POSITION**(  
 **POSITION\_ID NUMBER GENERATED AS IDENTITY PRIMARY KEY**,  
 **DESCRIPTION VARCHAR**(30) **NOT NULL**);  
  
**CREATE TABLE** EMPLOYEE(  
 **EMPLOYEE NUMBER GENERATED AS IDENTITY PRIMARY KEY**,  
 **PERSON\_ID NUMBER REFERENCES** PERSON(**PERSON\_ID**) **NOT NULL**,  
 **CONTRACT\_ID NUMBER NOT NULL**,  
 **POSITION\_ID NUMBER REFERENCES POSITION**(**POSITION\_ID**) **NOT NULL**,  
 **BRANCH\_ID NUMBER REFERENCES** BRANCH(**BRANCH\_ID**) **NOT NULL**,  
 **SALARY NUMBER NOT NULL**);  
  
**CREATE TABLE** COURSE(  
 **COURSE\_ID NUMBER GENERATED AS IDENTITY PRIMARY KEY**,  
 **LANGUAGE VARCHAR**(20) **NOT NULL**,  
 **BRANCH\_ID NUMBER REFERENCES** BRANCH(**BRANCH\_ID**) **NOT NULL**);  
  
  
**CREATE TABLE** STUDENT(  
 **STUDENT\_ID NUMBER GENERATED AS IDENTITY PRIMARY KEY**,  
 **PERSON\_ID NUMBER REFERENCES** PERSON(**PERSON\_ID**),  
 **COURSE\_ID NUMBER REFERENCES** COURSE(**COURSE\_ID**)  
);

# Приложение Б

Класс Data.StudentEntity:

**package** Data;  
  
**import** javax.persistence.\*;  
  
@Entity  
@Table(name=**"STUDENT"**)  
**public class** StudentEntity {  
  
 @Id  
 @Column(name=**"STUDENT\_ID"**)  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 **private** Long **id**;  
  
 @OneToOne  
 @JoinColumn(name = **"PERSON\_ID"**, referencedColumnName = **"PERSON\_ID"**)  
 **private** PersonEntity **person**;  
  
 @ManyToOne  
 @JoinColumn(name = **"COURSE\_ID"**, referencedColumnName = **"COURSE\_ID"**)  
 **private** CourseEntity **course**;  
  
 **public** StudentEntity() {  
 }  
  
 **public** Long getId() {  
 **return id**;  
 }  
  
 **public void** setId(Long id) {  
 **this**.**id** = id;  
 }  
  
 **public** PersonEntity getPerson() {  
 **return person**;  
 }  
  
 **public void** setPerson(PersonEntity person) {  
 **this**.**person** = person;  
 }  
  
 **public** CourseEntity getCourse() {  
 **return course**;  
 }  
  
 **public void** setCourse(CourseEntity course) {  
 **this**.**course** = course;  
 }  
}

Класс Data.GenericDataBaseAccess:

**package** Data;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
**import** javax.persistence.NoResultException;  
**import** javax.persistence.criteria.CriteriaQuery;  
**import** java.io.Serializable;  
**import** java.util.List;  
  
**public class** GenericDataBaseAccess<T, PK **extends** Serializable> {  
  
 **private** EntityManagerFactory **entityManagerFactory** = PersistenceManager.*getInstance*().getEntityManagerFactory();  
  
 **private** Class<T> **type**;  
  
 **public** GenericDataBaseAccess() {  
 }  
  
  
 **public** GenericDataBaseAccess(Class <T> type) {  
 **this**.**type**=type;  
 }  
  
 **protected** EntityManager getEntityManager() {  
 **return entityManagerFactory**.createEntityManager();  
 }  
  
 **public boolean** create(T entity)  
 {  
 EntityManager entityManager = getEntityManager();  
 entityManager.getTransaction().begin();  
 entityManager.persist(entity);  
 entityManager.getTransaction().commit();  
 entityManager.close();  
 **return true**;  
 }  
  
 **public void** update(T entity)  
 {  
 EntityManager entityManager = getEntityManager();  
 entityManager.getTransaction().begin();  
 **try** {  
 entityManager.merge(entity);  
 }  
 **catch** (IllegalStateException ignored)  
 {}  
 entityManager.getTransaction().commit();  
 entityManager.close();  
 }  
  
 **public** T findById(PK id)  
 {  
 EntityManager entityManager = getEntityManager();  
 T entity;  
 **try** {  
 entity = entityManager.find(**type**, id);  
 }  
 **catch** (NoResultException r)  
 {  
 entity = **null**;  
 }  
 entityManager.close();  
  
 **return** entity;  
 }  
  
 **public void** delete(T entity) {  
 EntityManager entityManager = getEntityManager();  
 entityManager.getTransaction().begin();  
 entityManager.remove(entityManager.contains(entity) ? entity : entityManager.merge(entity));  
 entityManager.getTransaction().commit();  
 entityManager.close();  
 }  
  
  
 **public** List<T> findAll() {  
 EntityManager entityManager = getEntityManager();  
 CriteriaQuery<T> criteria = entityManager.getCriteriaBuilder().createQuery(**type**);  
 criteria.select(criteria.from(**type**));  
 List<T> entities;  
 **try** {  
 entities = entityManager.createQuery(criteria).getResultList();  
 }  
 **catch** (NoResultException e)  
 {  
 entities = **null**;  
 }  
 entityManager.close();  
 **return** entities;  
 }  
  
 **public boolean** deleteAll() {  
 EntityManager entityManager = getEntityManager();  
 entityManager.getTransaction().begin();  
 **try** {  
 List<T> entityList = findAll();  
 **for** (T entity : entityList)  
 {  
 delete(entity);  
 }  
 }  
 **catch** (Exception e)  
 {  
 e.printStackTrace();  
 entityManager.getTransaction().rollback();  
 entityManager.close();  
 **return false**;  
 }  
 entityManager.getTransaction().commit();  
 entityManager.close();  
 **return true**;  
 }  
}

Класс View.StudentView:

**package** View;  
  
**import** Data.\*;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.Initializable;  
**import** javafx.scene.control.\*;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** javafx.scene.text.Text;  
  
**import** java.net.URL;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.ResourceBundle;  
  
**public class** StudentsView **implements** Initializable {  
  
 @FXML  
 **public** Text **school**;  
 @FXML  
 **public** Button **menuStudents**;  
 @FXML  
 **public** Button **menuGroups**;  
 @FXML  
 **public** Button **doAction**;  
 @FXML  
 **public** ComboBox<String> **action**;  
  
 @FXML  
 **public** TextField **name**;  
 @FXML  
 **public** TextField **surname**;  
 @FXML  
 **public** TextField **course**;  
 @FXML  
 **public** TableView<StudentsRaw> **studentsTable**;  
 @FXML  
 **public** Text **error**;  
  
 **private** GenericDataBaseAccess<StudentEntity, Long> **dbaStudent** = **new** GenericDataBaseAccess<>(StudentEntity.**class**);  
  
 @Override  
 **public void** initialize(URL location, ResourceBundle resources) {  
 TableColumn<StudentsRaw, Long> idColumn = **new** TableColumn<>(**"id"**);  
 TableColumn<StudentsRaw, String> nameColumn = **new** TableColumn<>(**"Имя"**);  
 TableColumn<StudentsRaw, String> surnameColumn = **new** TableColumn<>(**"Фамилия"**);  
 TableColumn<StudentsRaw, String> addressColumn = **new** TableColumn<>(**"Адрес занятий"**);  
 TableColumn<StudentsRaw, Long> courseColumn = **new** TableColumn<>(**"Группа"**);  
 TableColumn<StudentsRaw, String> languageColumn = **new** TableColumn<>(**"Язык"**);  
  
 idColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 nameColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"name"**));  
 surnameColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"surname"**));  
 addressColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"address"**));  
 courseColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"courseId"**));  
 languageColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"language"**));  
  
 idColumn.setPrefWidth(50);  
 nameColumn.setPrefWidth(120);  
 surnameColumn.setPrefWidth(124);  
 addressColumn.setPrefWidth(200);  
 courseColumn.setPrefWidth(50);  
 languageColumn.setPrefWidth(79);  
  
 **this**.**action**.setItems(FXCollections.*observableArrayList*(**"Добавить"**, **"Удалить"**));  
 **this**.**action**.setValue(**"Добавить"**);  
  
 **studentsTable**.getColumns().addAll(idColumn, nameColumn, surnameColumn, addressColumn, courseColumn, languageColumn);  
 updateTable();  
  
 }  
  
 @FXML  
 **private void** updateAction(){  
 **this**.**doAction**.setText(**this**.**action**.getValue());  
 **this**.**name**.setText(**""**);  
 **if** (**this**.**action**.getValue().equals(**"Удалить"**)){  
 **this**.**surname**.setVisible(**false**);  
 **this**.**course**.setVisible(**false**);  
  
 **this**.**name**.setPromptText(**"id студента"**);  
 }  
 **else** {  
 **this**.**surname**.setVisible(**true**);  
 **this**.**course**.setVisible(**true**);  
  
 **this**.**name**.setPromptText(**"Имя"**);  
 }  
  
 }  
  
 **private void** updateTable(){  
 List<StudentEntity> students = **dbaStudent**.findAll();  
 List<StudentsRaw> studentsRaws = **new** ArrayList<>();  
  
 **for**(StudentEntity student : students){  
 StudentsRaw studentRaw = **new** StudentsRaw();  
 studentRaw.setId(student.getId());  
 studentRaw.setName(student.getPerson().getName());  
 studentRaw.setSurname(student.getPerson().getSurname());  
 studentRaw.setCourseId(student.getCourse().getId());  
 studentRaw.setLanguage(student.getCourse().getLanguage());  
 BranchEntity branch = student.getCourse().getBranch();  
 studentRaw.setAddress(branch.getStreet() + **", "** + branch.getBuilding());  
  
 studentsRaws.add(studentRaw);  
 }  
 ObservableList<StudentsRaw> raws = FXCollections.*observableArrayList*(studentsRaws);  
 **studentsTable**.setItems(raws);  
 }  
  
 **public void** doAction(ActionEvent event) {  
 **this**.**error**.setVisible(**false**);  
 **if** (**this**.**doAction**.getText().equals(**"Удалить"**)){  
 **long** id;  
 StudentEntity student;  
 **try**{  
 id = Long.*parseLong*(**this**.**name**.getText());  
 }  
 **catch** (Exception e){  
 **this**.**error**.setVisible(**true**);  
 **return**;  
 }  
 student = **dbaStudent**.findById(id);  
 **if** (student == **null**){  
 **this**.**error**.setVisible(**true**);  
 **return**;  
 }  
 **dbaStudent**.delete(student);  
 }  
 **else**{  
 **if** (**this**.**name**.getText().isEmpty() || **this**.**name**.getText().equals(**""**)  
 || **this**.**surname**.getText().isEmpty() || **this**.**surname**.getText().equals(**""**)  
 || **this**.**course**.getText().isEmpty() || **this**.**course**.getText().equals(**""**)){  
 **this**.**error**.setVisible(**true**);  
 **return**;  
 }  
 **try**{  
 **long** group = Long.*parseLong*(**this**.**course**.getText());  
 GenericDataBaseAccess<CourseEntity, Long> dbaCourses = **new** GenericDataBaseAccess<>(CourseEntity.**class**);  
 GenericDataBaseAccess<PersonEntity, Long> dbaPerson = **new** GenericDataBaseAccess<>(PersonEntity.**class**);  
  
 CourseEntity course = dbaCourses.findById(group);  
 PersonEntity person = **new** PersonEntity();  
 person.setName(**this**.**name**.getText());  
 person.setSurname(**this**.**surname**.getText());  
  
 dbaPerson.create(person);  
  
 StudentEntity newStudent = **new** StudentEntity();  
 newStudent.setCourse(course);  
 newStudent.setPerson(person);  
  
 **dbaStudent**.create(newStudent);  
 }  
 **catch** (Exception e){  
 System.***out***.print(e.getMessage());  
 **this**.**error**.setVisible(**true**);  
 **return**;  
 }  
 }  
 updateTable();  
 }  
  
}

# Приложение В

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>

<?import javafx.scene.control.ComboBox?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.control.TableView?>

<?import javafx.scene.control.TextField?>

<?import javafx.scene.layout.Pane?>

<?import javafx.scene.text.Font?>

<?import javafx.scene.text.Text?>

<Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="600.0" prefWidth="800.0" style="-fx-background-color: rgb(255,255,234);" xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller="View.StudentsView">

<children>

<TableView fx:id="studentsTable" layoutX="173.0" layoutY="29.0" prefHeight="571.0" prefWidth="627.0" />

<ComboBox fx:id="action" layoutX="14.0" layoutY="154.0" onAction="#updateAction" prefWidth="150.0" />

<TextField fx:id="name" layoutX="15.0" layoutY="194.0" promptText="Имя" />

<TextField fx:id="surname" layoutX="15.0" layoutY="235.0" promptText="Фамилия" />

<TextField fx:id="course" layoutX="15.0" layoutY="275.0" promptText="id группы" />

<Button fx:id="doAction" layoutX="46.0" layoutY="318.0" mnemonicParsing="false" onAction="#doAction" prefHeight="17.0" prefWidth="86.0" text="Добавить">

<font>

<Font size="14.0" />

</font>

</Button>

<Text fx:id="error" fill="RED" layoutX="35.0" layoutY="381.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Некорректные данные" textAlignment="CENTER" visible="false" wrappingWidth="108.373046875">

<font>

<Font size="15.0" />

</font>

</Text>

<Label alignment="CENTER" contentDisplay="CENTER" layoutX="173.0" layoutY="1.0" prefHeight="30.0" prefWidth="627.0" text="Список студентов" textAlignment="CENTER" textFill="#4a4a4a">

<font>

<Font size="20.0" />

</font>

</Label>

<Text fx:id="error1" fill="#4a4a4a" layoutX="27.0" layoutY="47.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Школа изучения иностранных языков" textAlignment="CENTER" wrappingWidth="124.373046875">

<font>

<Font size="17.0" />

</font>

</Text>

</children>

</Pane>