МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

**Пояснительная записка к курсовой работе**

**по теме «Автоматизированная система управления данными родильного дома»**

по дисциплине

Управление данными

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Балашова Т. И.

(подпись)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Спиридонов Д. С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

17-АС

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород, 2020

**Содержание**

1. [Цель работы](#Цель_работы) 3
   1. [Основания и назначения разработки](#Основания_и_назначения_разработки) 3
   2. [Требования к структуре базы данных](#Требования_к_структуре_базы_данных) 3
2. [Проектирование базы данных](#Проектирование_базы_данных) 4
   1. [Описание предметной области](#Предметная_область) 4
   2. [Описание отношений](#Описание_отношений) 5
   3. [ER диаграмма](#ER_диаграмма) 7
   4. [Логическая схема базы данных](#Логическая_схема_базы_данных) 7
3. [Создание программы](#Создание_программы) 9
   1. [Разработка базы данных в среде PostgreSQL](#Разработка_базы_данных) 9
   2. [Разработка программы в IntellIJ IDEA](#Разработка_программы_в_IntellIJ_IDEA) 11
      1. [**Связь базы данных с проектом**](#Связь_базы_данных_с_проектом)**...............................................................11**
      2. [**Визуализация проекта**](#Визуализация_проекта)**..............................................................................13**
4. [Руководство пользователя](#Руководство_пользователя) 14
5. [Руководство программиста](#Руководство_программиста) 18
   1. [Минимальные требования к клиентскому ПК](#Минимальные_требования_к_ПК) 18
   2. [Описание структуры проекта](#Описание_структуры_проекта) 18
6. [Заключение](#Заключение) 20
7. [Литература](#Литература) 21

[Приложение](#Приложение)

**1. Цель работы**

**1.1. Основания и назначения разработки**

Создать информационную систему “Автоматизированная система управления данными родильного дома”, предназначенную для ввода, редактирования и использования информации о врачах, палатах, детях и их матерях.

**1.2. Требования к структуре базы данных**

База данных должна хранить следующую информацию:

* **Врачи:** полное имя, специальность, дата рождения
* **Дети:** полное имя, дата рождения
* **Матери:** полное имя, дата рождения
* **Палаты:** номер палаты, vip-статус, число мест

Данный проект должен обеспечить удобства в работе с базой данных для конечного пользователя. Для этого следует реализовать все необходимые операции над информацией в базе, такие как: добавление, удаление, редактирование и поиск.

**2. Проектирование базы данных**

**2.1. Описание предметной области**

Предметная область информационной системы - это материальная система или система, характеризующая элементы материального мира, информация о которой хранится и обрабатывается. Предметная область рассматривается как некоторая совокупность реальных объектов и связей между ними.

Для успешного и эффективного функционирования всех без исключения функциональных типов медицинских учреждений (клиник, родильных домов, стационаров, госпиталей и др.) требуются автоматизированные системы для формирования базы данных на основе имеющейся в данном учреждении информации.

Одним из типов медицинских учреждений, очевидно требующих автоматизации, является родильный дом.

Родильный дом — это медицинское учреждение, обеспечивающее квалифицированную медицинскую помощь женщинам во время беременности, непосредственно в ходе родов и в послеродовой период, а также медицинскую помощь новорождённым.

Основная цель существующей информационной системы родильного дома - обеспечение хранения, обработки и предоставления информации о деятельности учреждения.

Рассмотрим следующий пример:

Девушка обращается в родильный дом, где на нее заводится карточка. В данной карточке хранится вся информация о пациентке. Для консультации ей нужно пройти в регистратуру, отстоять очередь, забрать карточку и только после этого проследовать к врачу, а позже сдать ее обратно в регистратуру, что тратит много времени и сил.

В момент родов с девушкой находятся врачи, акушеры, мед. сестры и т.д. Отчетность о наличии персонала на родах должна быть задокументирована. Также матерям выделяют места в палатах.

Имея автоматизированную базу данных, можно с легкостью оперировать такими данными.

Для достижения цели должны быть решены следующие задачи:

1. Хранение данных о врачах
2. Хранение данных о детях
3. Хранение данных о матерях
4. Хранение данных о палатах
5. Добавление информации
6. Поиск информации
7. Редактирование информации
8. Удаление информации

**2.2. Описание отношений**

Связь — это ассоциации (отношения) между сущностями в предметной области. Связи представляют собой соединения между частями БД.

Связи позволяют по одной сущности находить другие сущности, связанные с нею.

Каждая связь может иметь один из следующих типов:

* один-к-одному (1:1);
* один-ко-многим (1:М);
* многие-ко-многим (М:М).

Связь один-к-одному (1:1) означает, что один экземпляр первой сущности связан с одним экземпляром второй сущности. Связь один-к-одному чаще всего свидетельствует о том, что на самом деле мы имеем всего одну сущность, неправильно разделенную на две.

Связь один-ко-многим (1:М) означает, что один экземпляр первой сущности связан с несколькими экземплярами второй сущности.

Связь многие-ко-многим (М:М) означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности. Тип связи много-ко-многим является временным типом связи, допустимым на ранних этапах разработки модели. В дальнейшем этот тип связи должен быть заменен двумя связями типа один-ко-многим путем создания промежуточной сущности.

Основываясь на теоретическом материале, необходимо проанализировать связи между сущностями родильного дома.

Мед. персонал помогает родиться ребенку, также за свою практику мед. работник взаимодействует с несколькими детьми, поэтому необходима связь многие-ко-многим (M:M).

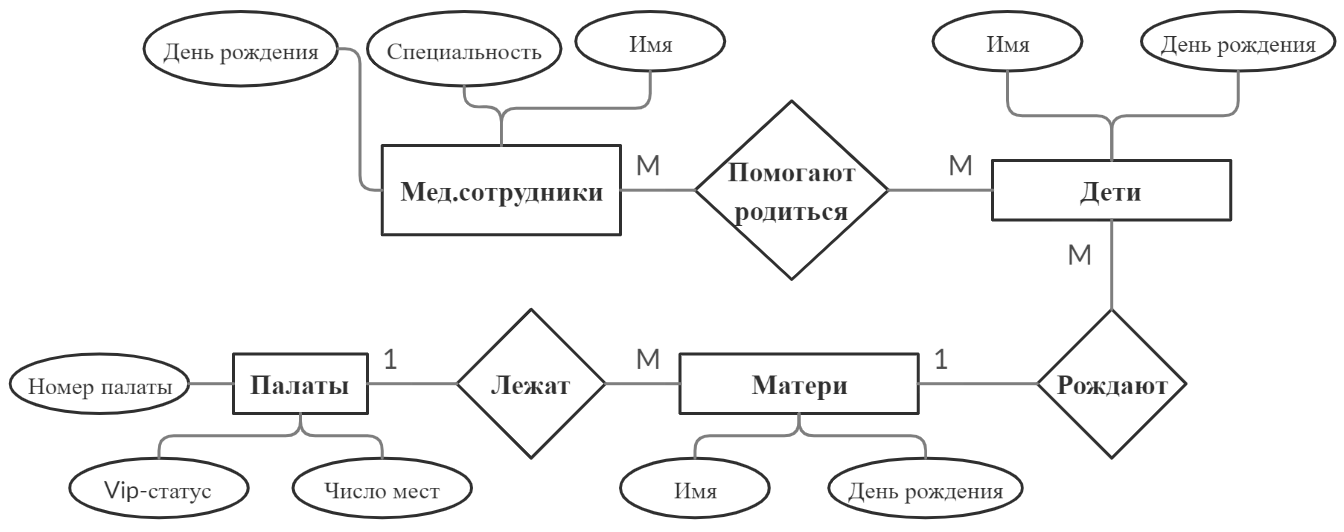
У ребенка может быть только одна биологическая мать, но мать может родить несколько детей, поэтому необходима связь один-ко-многим (1:М).

В палате могут находиться несколько матерей, но мать может лежать только в одной палате, поэтому необходима связь один-ко-многим (1:М).

**2.3. ER диаграмма**

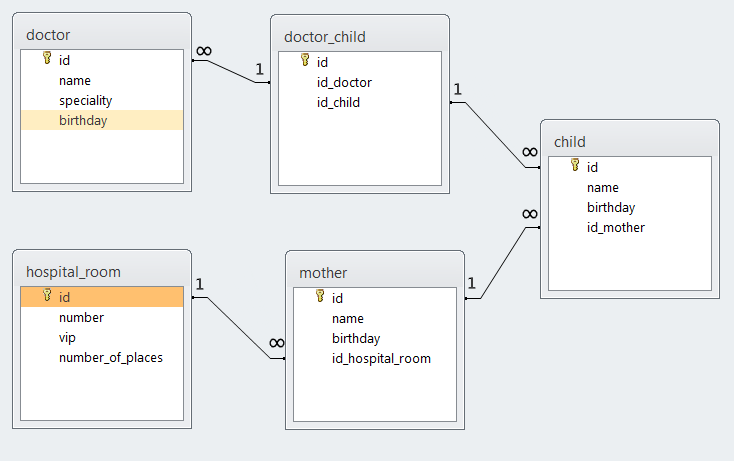
Все ER-диаграммы используют графическое изображение сущностей предметной области, их свойств (атрибутов) и взаимосвязей между сущностями.

Исходя из предыдущего подпункта, необходимо построить ER диаграмму, удовлетворяющую запросам родильного дома.



**2.4.** **Логическая схема базы данных**

Следующим шагом проектирования является создание логической структуры реляционной базы данных. Каждый информационный объект модели данных отображается с соответствующей реляционной таблицей. Структура реляционной таблицы определяется требуемым составом соответствующего информационного объекта, где каждый столбец (поле) соответствует одному из реквизитов объекта. Ключевые реквизиты объекта образуют уникальный ключ реляционной таблицы. Для каждого столбца указывается формат и размер данных. Строки (записи) таблицы соответствуют экземплярам объекта и генерируются при загрузке таблицы.



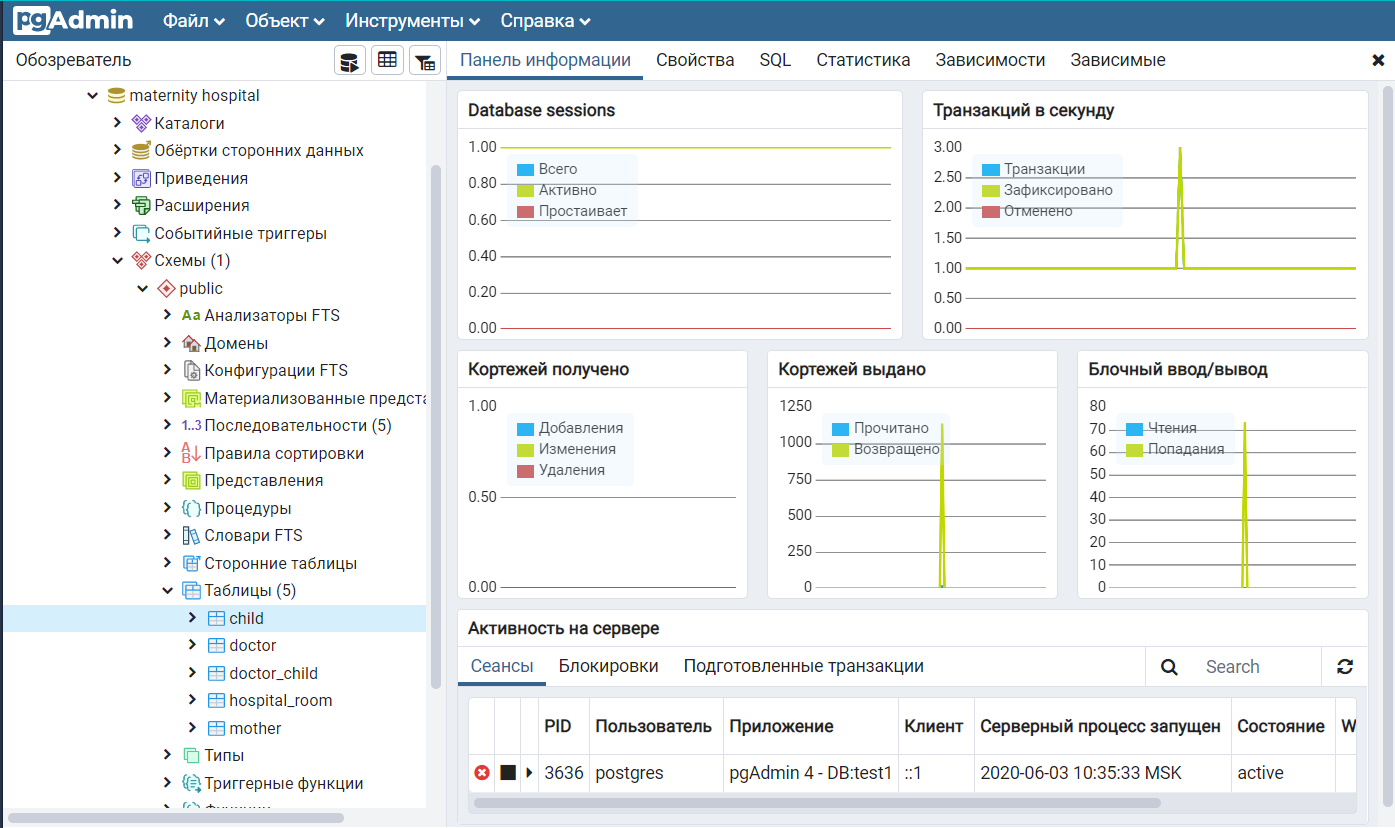
Как уже сказано в подпункте 3.2., тип связи многие-ко-многим (doctor↔child) является временным типом связи, допустимым на ранних этапах разработки модели, и необходимо заменить ее на две связи один-ко-многим путем добавления новой сущности (doctor\_child).

**3.** **Создание программы**

**3.1.** **Разработка базы данных в среде PostgreSQL**

Для создания базы данных была выбрана среда PostgreSQL, так как был опыт работы с программой pgAdmin 4.

Внешний интерфейс программы представлен ниже.



В данной программе была создана новая база данных «moternity hospital» с сущностями: «doctor», «doctor\_child», «child», «mother», «hospital\_room».

Таблица doctor:

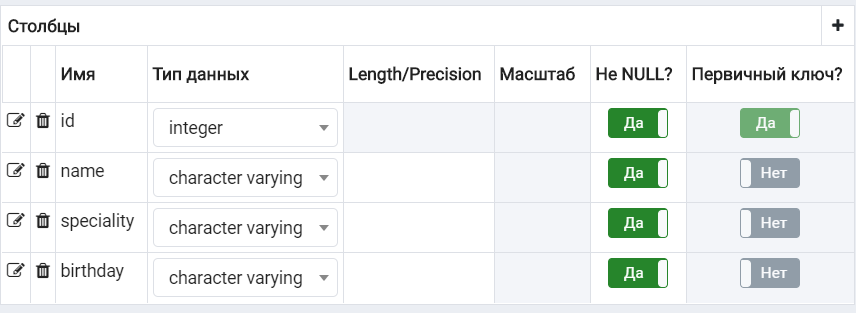


Таблица doctor\_child:



Таблица child:

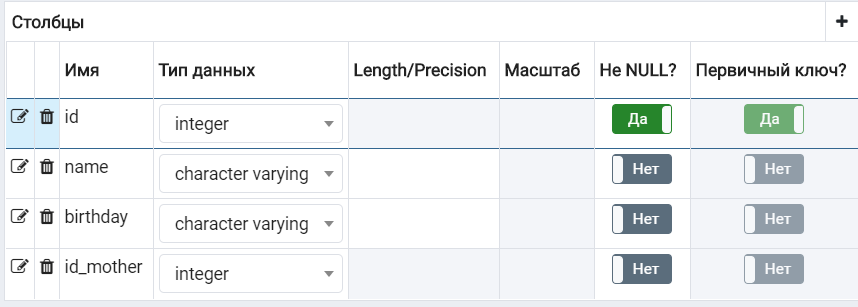


Таблица mother:

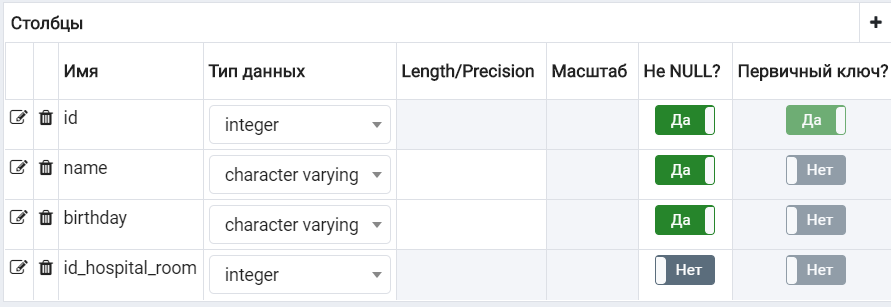
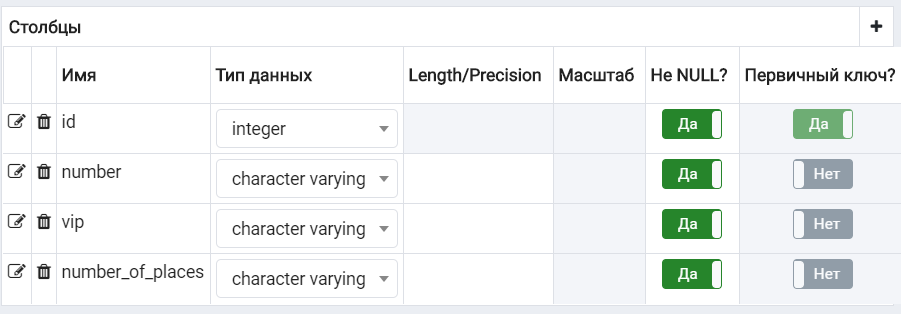


Таблица hospital\_room:



Для генерации последовательностей «id» была использована последовательность SEQUENCE.

**3.2.** **Разработка программы в IntellIJ IDEA**

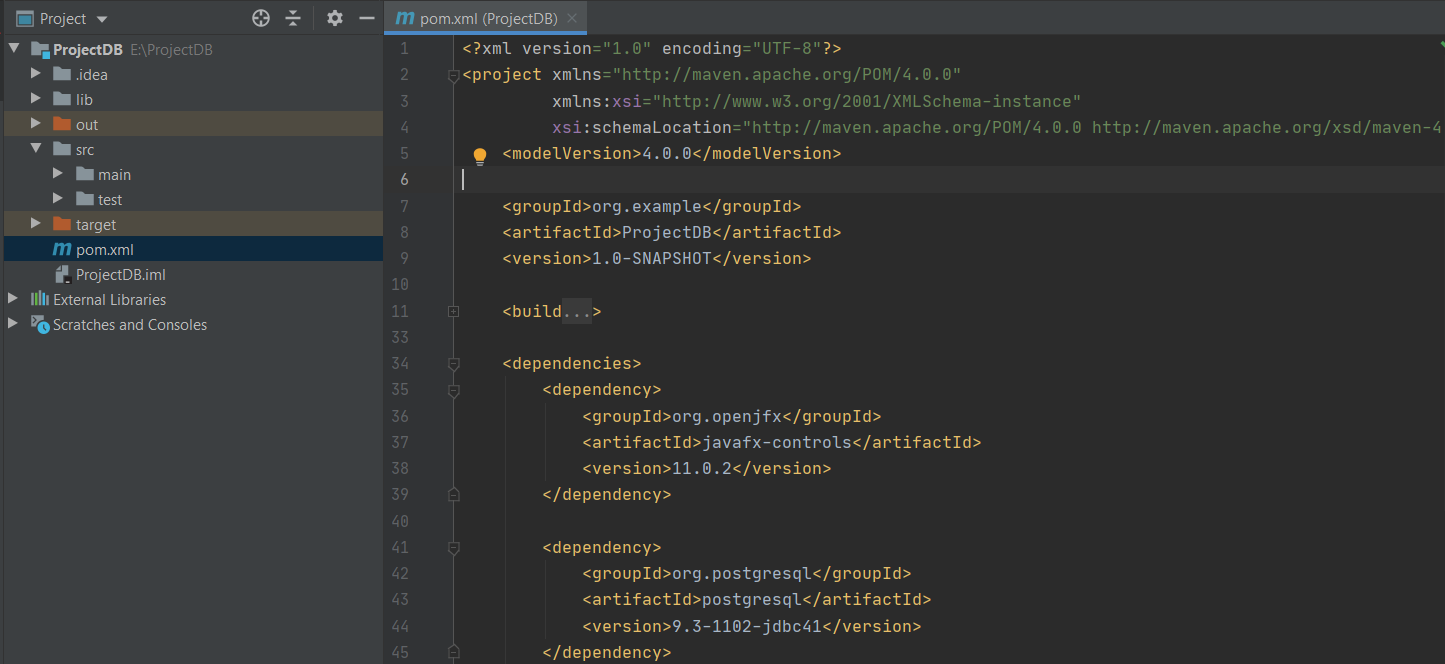
IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Данная среда разработки и язык программирования Java были выбраны по той причине, что у меня имеется опыт работы с ними.

**3.2.1.** **Связь базы данных с проектом**

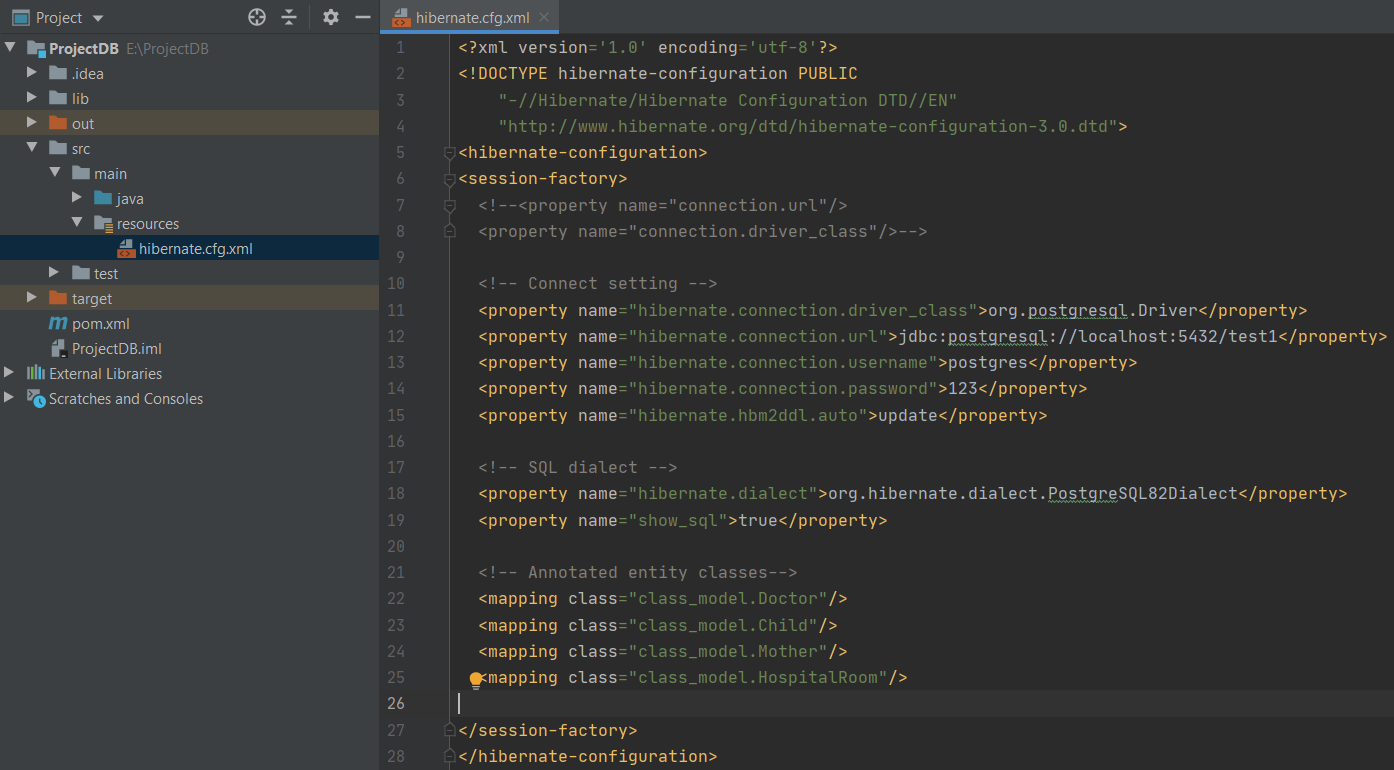
Для связи базы данных в PostgreSQL с проектом в IntellIJ IDEA было принято решение добавить фреймворк Maven и библиотеку Hibernate.

При добавлении Maven создается файл pom.xml, в котором прописывается сборка проекта.

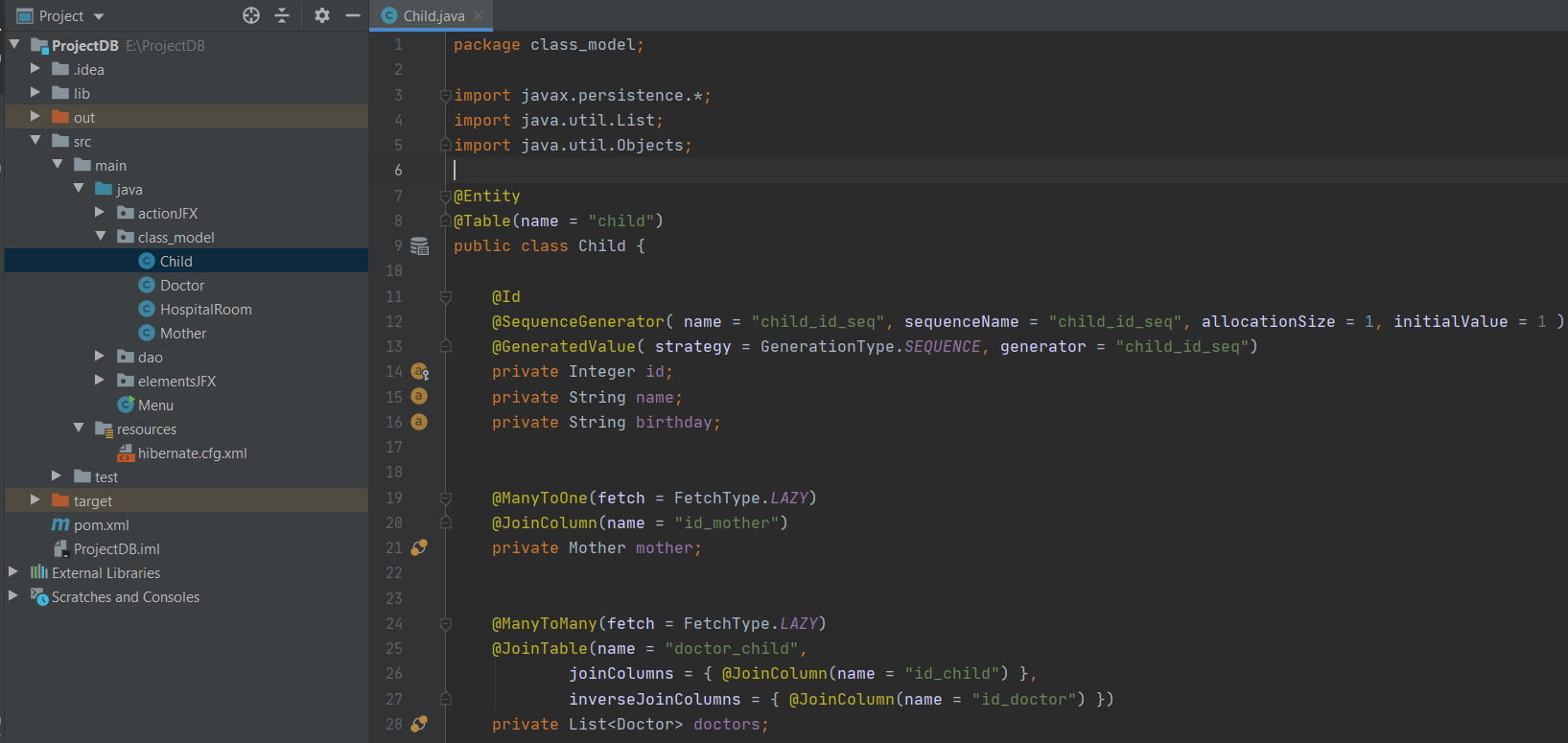


(представлена лишь часть кода)

При добавлении Hibernate создается файл hibernate.cfg.xml, в котором прописывается подключение к серверу PostgreSQL и непосредственно к базе данных.

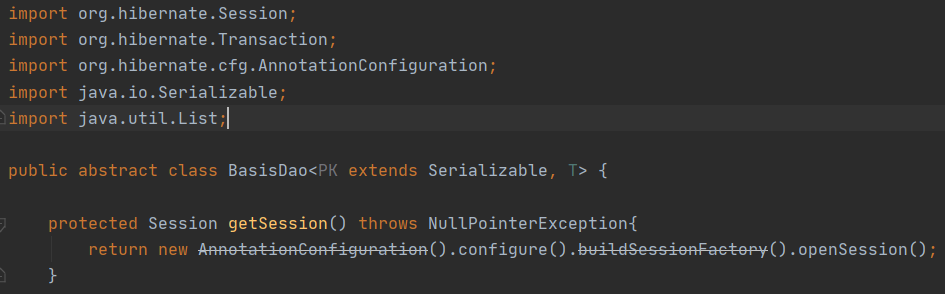


Связь таблиц осуществляется с классами проекта:



(для ознакомления подключения сущности к классу представлена лишь часть кода)

Для операций над базой был использован интерфейс Session библиотеки Hibernate.



**3.2.2.** **Визуализация проекта**

Для создание пользовательского интерфейса необходимо подключить инструмент JavaFX.

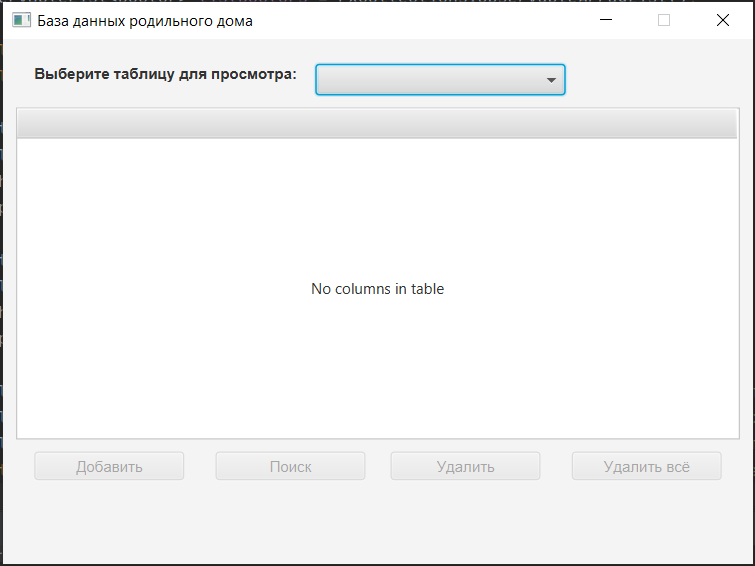
JavaFX представляет инструментарий для создания кроссплатформенных графических приложений на платформе Java.

JavaFX позволяет создавать приложения с богатой насыщенной графикой благодаря использованию аппаратного ускорения графики и возможностей GPU.

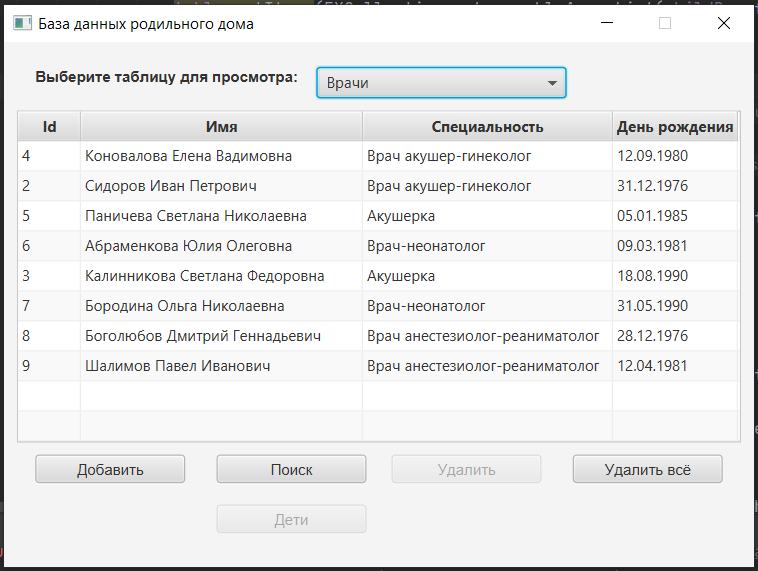
JavaFX предоставляет большие возможности по сравнению с рядом других подобных платформ, в частности, по сравнению со Swing. Это и большой набор элементов управления, и возможности по работе с мультимедиа, двухмерной и трехмерной графикой, декларативный способ описания интерфейса с помощью языка разметки FXML, возможность стилизации интерфейса с помощью CSS, интеграция со Swing и многое другое.

**4.** **Руководство пользователя**

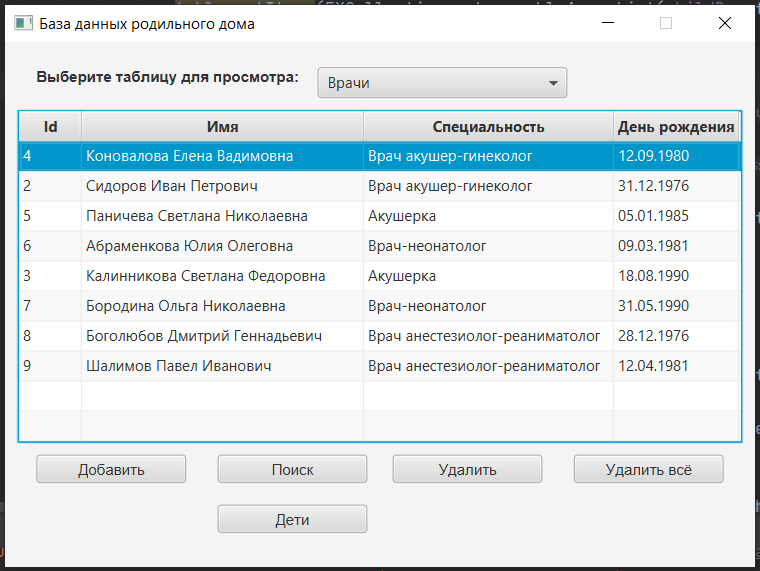
При запуске программы перед пользователем открывается меню. На данном этапе все кнопки находятся в нерабочем состоянии.



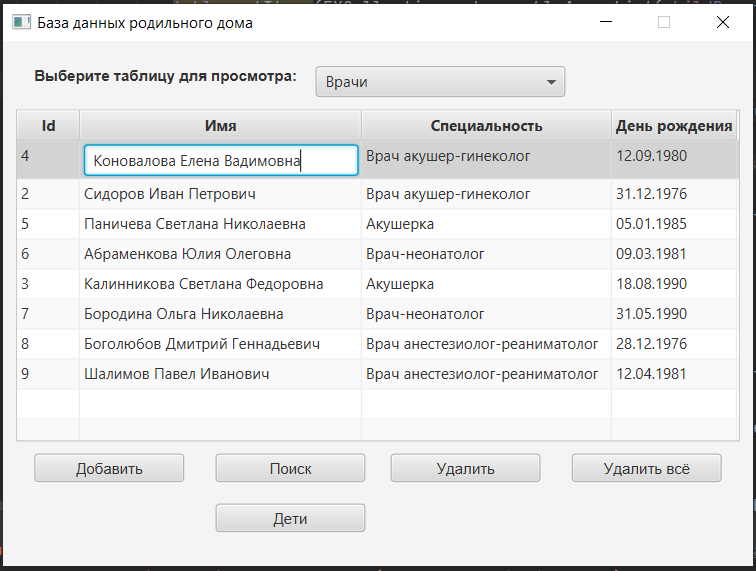
Далее следует выбрать таблицу для просмотра (врачи, дети, матери или палаты). После этого появится возможность использования кнопок: «добавить», «поиск», «удалить всё».



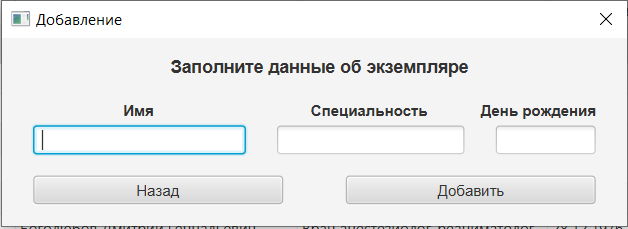
При одиночном нажатии на экземпляр активируется кнопка «удалить», а также кнопка(-и) связи (для каждой таблицы своя(-и)).



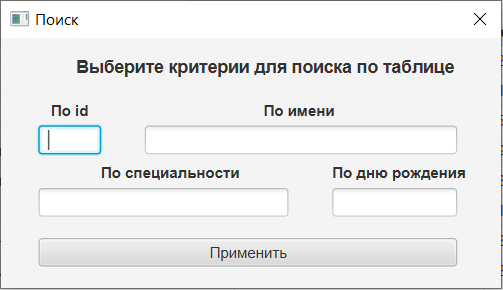
При двойном нажатии левой кнопкой мыши на атрибут экземпляра появится возможность редактировать выбранное поле. После окончания редактирования нужно нажать на кнопку «Enter».



При нажатии на кнопку «добавить» откроется окно заполнения экземпляра.

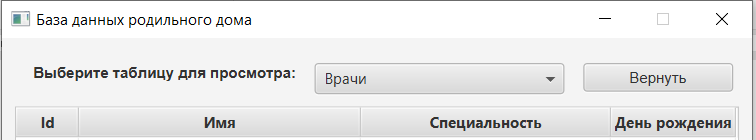


При нажатии на кнопку «поиск» откроется окно поиска. Искать экземпляр(-ы) можно по любому атрибуту, также их можно комбинировать. В случае отсутствия данного экземпляра таблица будет пуста.

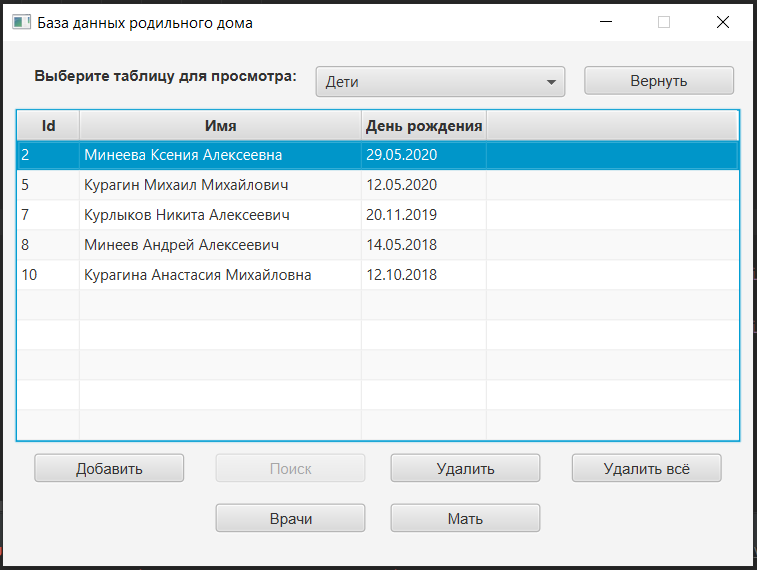


После задания фильтра необходимо нажать кнопку «применить». В таблице появятся отфильтрованные экземпляры. Для выхода из окна поиска без фильтрации нужно нажать на кнопку «применить», но все поля должны быть пустые.

Также в правом верхнем углу появится кнопка «вернуть». Ее необходимо нажать после завершения работы с поисковиком.



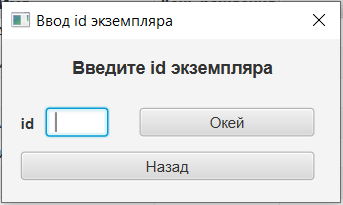
У каждой таблицы есть своя кнопка(-и) связи. При нажатии на нее появится соответствующая таблица с привязанными к данному экземпляру экземплярами.



(нажата кнопка «дети» в таблице «врачи»)

В данном окне все кнопки работают по тому же принципу, что и ранее, но уже изменяя связь между экземплярами:

* «Добавить» - добавляет связь между экземплярами (открывается окно, в которое нужно записать «id» экземпляра, которого нужно привязать).



* «Удалить» - удаляет связь между экземплярами.
* «Удалить всё» - удаляет все связи с выбранным экземпляром.
* «Вернуть» - возвращает обратно таблицы базы данных.

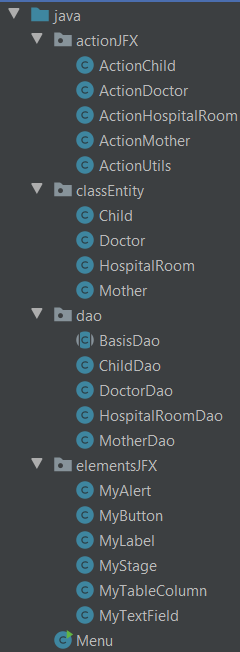
**Исключение:** изменение атрибутов экземпляра сохраняется в базе.

**5.** **Руководство программиста**

**5.1.** **Минимальные требования к клиентскому ПК**

* компьютер с процессором Intel Pentium и выше или совместимый
* 256 МБ RAM (не считая памяти выделенной под ОС)
* Монитор
* Клавиатура
* Мышь
* операционная система MS Windows® 7-10
* NET Framework 4.0

**5.2.** **Описание структуры проекта**

В проекте содержится 4 пакета классов и стартовый класс Menu.

* Package classEntity содержит модели данных, т.е. классы с привязкой к сущностям базы данных. В них записаны связи между таблицами и атрибуты таблиц.
* Package dao (Data Access Object) содержит классы перенаправления запросов к базе данных и абстрактный класс BasisDao. Перенаправление идет к классу BasisDao. Данный класс отвечает за доступ к данным базы (добавление, удаление, поиск и т.д.).
* Package elementsJFS содержит классы с конструкторами и базовыми методами элементов JavaFX. Данные классы служат для компоновки настроек элементов в конструкторах.
* Package actionJFX содержит классы с обработкой запросов к базе данных из пользовательского интерфейса. Пользователь выбирает действие над данными, а данные классы обрабатывают этот запрос. В случае успешного выполнения всех условий запроса он отправляется на исполнение в dao.
* Класс Menu является стартовым, с него запускается программа. Он является средством взаимодействия пользователя с программой с помощью таблицы и кнопок.

**6.** **Заключение**

В результате выполнения проекта была создана информационная система “Автоматизированная система управления данными родильного дома”. В процессе тестирования программы ошибок не обнаружено.

Цель данной работы достигнута. В ходе работы была проанализирована научная и методическая литература по проблеме разработки программы, была разработана наиболее подходящая база данных в среде PosgreSQL, а также был спроектирован интерфейс программы и порядок ее взаимодействия с пользователем в среде IntellIJ IDEA.

Таким образом, с помощью данной программы обеспечивается удобство в работе с базой данных для конечного пользователя.

**7.** **Литература**

1. Кириллов В.В. Введение в реляционные базы данных / В. В. Кириллов, Г. Ю. Громов — СПб.: БХВ-Петербург, 2009. — 464 с.
2. Кузин, А. В. Базы данных: учеб. пособ. для студ. высш. учеб. заведений / А. В. Кузин, С. В. Левонисова. — 2-е изд. — М.: Издательский центр «Академия», 2008. — 320 с.
3. Машнин Т.С. JavaFX 2.0: разработка RIA-приложений. – СПб.: БХВ-Петербург, 2012. – 320 с.
4. Моргунов, Е. П. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. — СПб.: БХВ-Петербург, 2018. — 336 с.
5. Шилдт, Г. Java 8: руководство для начинающих, 6-е изд.: Пер. с англ. - М. ООО "И.Д. Вильяме", 2015. - 720 с.

**Приложение**

**pom.xml**

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

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>

<artifactId>ProjectDB</artifactId>

<version>1.0-SNAPSHOT</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>8</source>

<target>8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.openjfx</groupId>

<artifactId>javafx-maven-plugin</artifactId>

<version>0.0.3</version>

<configuration>

<mainClass>Menu</mainClass>

</configuration>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>org.openjfx</groupId>

<artifactId>javafx-controls</artifactId>

<version>11.0.2</version>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>9.3-1102-jdbc41</version>

</dependency>

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.4.0-b180830.0359</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.7.Final</version>

</dependency>

<dependency>

<groupId>org.hibernate.javax.persistence</groupId>

<artifactId>hibernate-jpa-2.1-api</artifactId>

<version>1.0.0.Final</version>

</dependency>

</dependencies>

</project>

**hibernate.cfg.xml**

<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Connect setting -->

<property name="hibernate.connection.driver\_class">org.postgresql.Driver</property>

<property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/maternity\_hospital</property>

<property name="hibernate.connection.username">postgres</property>

<property name="hibernate.connection.password">123</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- SQL dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.PostgreSQL82Dialect</property>

<property name="show\_sql">true</property>

<!-- Annotated entity classes-->

<mapping class="classEntity.Doctor"/>

<mapping class="classEntity.Child"/>

<mapping class="classEntity.Mother"/>

<mapping class="classEntity.HospitalRoom"/>

</session-factory>

</hibernate-configuration>

**Menu.java**

import actionJFX.\*;

import classEntity.\*;

import dao.\*;

import elementsJFX.\*;

import javafx.application.Application;

import javafx.beans.value.ChangeListener;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.ComboBox;

import javafx.scene.control.TableView;

import javafx.scene.layout.Pane;

import javafx.scene.layout.VBox;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

public class Menu extends Application {

ObservableList<Child> listChilds = FXCollections.observableArrayList();

ObservableList<Mother> listMothers = FXCollections.observableArrayList();

ObservableList<HospitalRoom> listHospitalRoom = FXCollections.observableArrayList();

ObservableList<Doctor> listDoctors = FXCollections.observableArrayList();

boolean flagFind = false;

boolean flagFindDaughters = false;

Doctor doctor = new Doctor();

Child child = new Child();

Mother mother = new Mother();

HospitalRoom hospitalRoom = new HospitalRoom();

DoctorDao doctorDao = new DoctorDao();

ChildDao childDao = new ChildDao();

MotherDao motherDao = new MotherDao();

HospitalRoomDao hospitalRoomDao = new HospitalRoomDao();

MyAlert numberFormat = new MyAlert(Alert.AlertType.ERROR,"Здесь могут содержаться только цифры");

MyAlert vipFormat = new MyAlert(Alert.AlertType.ERROR,"Некорректно введен Vip-статус! (+/-)");

public static void main(String[] args) {

DoctorDao doctorDao = new DoctorDao();

doctorDao.findAll();

Application.launch(args);

}

public void reset(){

this.flagFind = false;

this.flagFindDaughters = false;

this.doctor = null;

this.child = null;

this.mother = null;

this.hospitalRoom = null;

if (!listDoctors.isEmpty())

listDoctors.clear();

if (!listChilds.isEmpty())

listChilds.clear();

if (!listMothers.isEmpty())

listMothers.clear();

if (!listHospitalRoom.isEmpty())

listHospitalRoom.clear();

if (!listDoctors.isEmpty())

listDoctors.clear();

}

@Override

public void start(Stage stage) {

MyLabel lbEnterTable = new MyLabel("Выберите таблицу для просмотра:",12,25,20);

MyButton btnAdd = new MyButton("Добавить",12,120,25,330);

MyButton btnFind = new MyButton("Поиск",12,120,170,330);

MyButton btnDelete = new MyButton("Удалить",12,120,310,330);

MyButton btnDeleteAll = new MyButton("Удалить всё",12,120,455,330);

MyButton btnReset = new MyButton("Вернуть",12,120,465,20);

MyButton btnDoctors = new MyButton("Врачи",12,120,170,370);

MyButton btnMother = new MyButton("Мать",12,120,310,370);

MyButton btnChilds = new MyButton("Дети",12,120,170,370);

MyButton btnHospitalRoom = new MyButton("Палата",12,120,310,370);

MyButton btnMothers = new MyButton("Матери",12,120,170,370);

btnAdd.setDisable(true);

btnDelete.setDisable(true);

btnDeleteAll.setDisable(true);

btnFind.setDisable(true);

btnReset.setWork(false);

btnDoctors.setWork(false);

btnChilds.setWork(false);

btnMother.setWork(false);

btnMothers.setWork(false);

btnHospitalRoom.setWork(false);

TableView table = new TableView<>();

table.setPrefWidth(600);

table.setPrefHeight(270);

table.setEditable(true);

TableView.TableViewSelectionModel selectionModel = table.getSelectionModel();

ComboBox<String> langsComboBox = new ComboBox(FXCollections.observableArrayList("Врачи", "Дети", "Матери", "Палаты"));

langsComboBox.setLayoutX(250);

langsComboBox.setLayoutY(20);

langsComboBox.setPrefHeight(20);

langsComboBox.setPrefWidth(200);

langsComboBox.setOnAction(e -> {

table.getColumns().clear();

btnFind.setDisable(false);

btnDeleteAll.setDisable(false);

//Настройка кнопок

switch (langsComboBox.getValue()) {

case "Врачи": {

btnMother.setWork(false);

btnMothers.setWork(false);

btnHospitalRoom.setWork(false);

btnDoctors.setWork(false);

btnChilds.setWork(true);

btnDelete.setDisable(true);

btnChilds.setDisable(true);

btnReset.setWork(flagFind);

btnAdd.setDisable(flagFind);

if (flagFindDaughters) {

btnReset.setWork(true);

btnAdd.setWork(true);

btnFind.setDisable(true);

}

btnDeleteAll.btn.setOnAction(event -> {

ActionDoctor.deleteAll(flagFind,flagFindDaughters,listDoctors,child);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Врачи");

});

btnAdd.btn.setOnAction(event -> {

ActionDoctor.add(stage,listDoctors,flagFindDaughters,child);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Врачи");

});

btnFind.btn.setOnAction(event -> {

flagFind = ActionDoctor.find(stage,listDoctors);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Врачи");

});

btnReset.btn.setOnAction(event -> {

reset();

langsComboBox.setValue("Дети");

langsComboBox.setValue("Врачи");

});

selectionModel.selectedItemProperty().addListener((ChangeListener<Doctor>)(exemplar, oldVal, newVal) -> {

btnDelete.setDisable(exemplar.getValue() == null);

btnChilds.setDisable(exemplar.getValue() == null);

btnDelete.btn.setOnAction(event -> {

ActionDoctor.delete(flagFind,flagFindDaughters,listDoctors,exemplar.getValue(),child);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Врачи");

});

btnChilds.btn.setOnAction(event -> {

flagFindDaughters = true;

doctor = exemplar.getValue();

child = null;

mother = null;

hospitalRoom = null;

listChilds = FXCollections.observableArrayList(doctorDao.findChilds(doctor));

langsComboBox.setValue("Дети");

});

});

break;

}

case "Дети": {

btnMother.setWork(true);

btnMothers.setWork(false);

btnHospitalRoom.setWork(false);

btnDoctors.setWork(true);

btnChilds.setWork(false);

btnDelete.setDisable(true);

btnDoctors.setDisable(true);

btnMother.setDisable(true);

btnReset.setWork(flagFind);

btnAdd.setDisable(flagFind);

if (flagFindDaughters) {

btnReset.setWork(true);

btnAdd.setWork(true);

btnFind.setDisable(true);

}

btnDeleteAll.btn.setOnAction(event -> {

ActionChild.deleteAll(flagFind,flagFindDaughters,listChilds,doctor,mother);

langsComboBox.setValue("Матери");

langsComboBox.setValue("Дети");

});

btnAdd.btn.setOnAction(event -> {

ActionChild.add(stage,listChilds,flagFindDaughters,doctor,mother);

langsComboBox.setValue("Матери");

langsComboBox.setValue("Дети");

});

btnFind.btn.setOnAction(event -> {

flagFind = ActionChild.find(stage, listChilds);

langsComboBox.setValue("Матери");

langsComboBox.setValue("Дети");

});

btnReset.btn.setOnAction(event -> {

reset();

langsComboBox.setValue("Матери");

langsComboBox.setValue("Дети");

});

selectionModel.selectedItemProperty().addListener((ChangeListener<Child>)(exemplar, oldVal, newVal) -> {

btnDelete.setDisable(exemplar.getValue() == null);

btnDoctors.setDisable(exemplar.getValue() == null);

btnMother.setDisable(exemplar.getValue() == null);

btnDelete.btn.setOnAction(event -> {

ActionChild.delete(flagFind,flagFindDaughters,listChilds,exemplar.getValue(),doctor,mother);

langsComboBox.setValue("Матери");

langsComboBox.setValue("Дети");

});

btnDoctors.btn.setOnAction(event -> {

flagFindDaughters = true;

child = exemplar.getValue();

mother = null;

hospitalRoom = null;

doctor = null;

listDoctors = FXCollections.observableArrayList(childDao.findDoctors(child));

langsComboBox.setValue("Врачи");

});

btnMother.btn.setOnAction(event -> {

flagFindDaughters = true;

child = exemplar.getValue();

mother = null;

hospitalRoom = null;

doctor = null;

listMothers = FXCollections.observableArrayList(childDao.findMother(child));

langsComboBox.setValue("Матери");

});

});

break;

}

case "Матери": {

btnMother.setWork(false);

btnMothers.setWork(false);

btnHospitalRoom.setWork(true);

btnDoctors.setWork(false);

btnChilds.setWork(true);

btnDelete.setDisable(true);

btnHospitalRoom.setDisable(true);

btnChilds.setDisable(true);

btnReset.setWork(flagFind);

btnAdd.setDisable(flagFind);

if (flagFindDaughters) {

btnReset.setWork(true);

btnAdd.setWork(true);

btnFind.setDisable(true);

}

btnDeleteAll.btn.setOnAction(event -> {

ActionMother.deleteAll(flagFind,flagFindDaughters,listMothers,child,hospitalRoom);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Матери");

});

btnAdd.btn.setOnAction(event -> {

ActionMother.add(stage,listMothers,flagFindDaughters,child,hospitalRoom);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Матери");

});

btnFind.btn.setOnAction(event -> {

flagFind = ActionMother.find(stage,listMothers);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Матери");

});

btnReset.btn.setOnAction(event -> {

reset();

langsComboBox.setValue("Дети");

langsComboBox.setValue("Матери");

});

selectionModel.selectedItemProperty().addListener((ChangeListener<Mother>)(exemplar, oldVal, newVal) -> {

btnDelete.setDisable(exemplar.getValue() == null);

btnChilds.setDisable(exemplar.getValue() == null);

btnHospitalRoom.setDisable(exemplar.getValue() == null);

btnDelete.btn.setOnAction(event -> {

ActionMother.delete(flagFind,flagFindDaughters,listMothers,mother,child,hospitalRoom);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Матери");

});

btnChilds.btn.setOnAction(event -> {

flagFindDaughters = true;

mother = exemplar.getValue();

child = null;

hospitalRoom = null;

doctor = null;

listChilds = FXCollections.observableArrayList(motherDao.findChilds(mother));

langsComboBox.setValue("Дети");

});

btnHospitalRoom.btn.setOnAction(event -> {

flagFindDaughters = true;

mother = exemplar.getValue();

child = null;

hospitalRoom = null;

doctor = null;

listHospitalRoom = FXCollections.observableArrayList(motherDao.findHospitalRoom(mother));

langsComboBox.setValue("Палаты");

});

});

break;

}

case "Палаты": {

btnMother.setWork(false);

btnMothers.setWork(true);

btnHospitalRoom.setWork(false);

btnDoctors.setWork(false);

btnChilds.setWork(false);

btnDelete.setDisable(true);

btnMothers.setDisable(true);

btnReset.setWork(flagFind);

btnAdd.setDisable(flagFind);

if (flagFindDaughters) {

btnReset.setWork(true);

btnAdd.setWork(true);

btnFind.setDisable(true);

}

btnDeleteAll.btn.setOnAction(event -> {

ActionHospitalRoom.deleteAll(flagFind,flagFindDaughters,listHospitalRoom,mother);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Палаты");

});

btnAdd.btn.setOnAction(event -> {

ActionHospitalRoom.add(stage,listHospitalRoom,flagFindDaughters,mother);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Палаты");

});

btnFind.btn.setOnAction(event -> {

flagFind =ActionHospitalRoom.find(stage,listHospitalRoom);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Палаты");

});

btnReset.btn.setOnAction(event -> {

reset();

langsComboBox.setValue("Дети");

langsComboBox.setValue("Палаты");

});

selectionModel.selectedItemProperty().addListener((ChangeListener<HospitalRoom>)(exemplar, oldVal, newVal) -> {

btnDelete.setDisable(exemplar.getValue() == null);

btnMothers.setDisable(exemplar.getValue() == null);

btnDelete.btn.setOnAction(event -> {

ActionHospitalRoom.delete(flagFind,flagFindDaughters,listHospitalRoom,exemplar.getValue(),mother);

langsComboBox.setValue("Дети");

langsComboBox.setValue("Палаты");

});

btnMothers.btn.setOnAction(event -> {

flagFindDaughters = true;

hospitalRoom = exemplar.getValue();

child = null;

mother = null;

doctor = null;

listMothers = FXCollections.observableArrayList(hospitalRoomDao.findMothers(hospitalRoom));

langsComboBox.setValue("Матери");

});

});

break;

}

}

//Настройка таблицы

switch (langsComboBox.getValue()) {

case "Врачи": {

MyTableColumn idCol = new MyTableColumn("Id","id",50,false);

MyTableColumn nameCol = new MyTableColumn("Имя","name",200,true);

nameCol.tableColumn.setOnEditCommit(event -> {

Doctor doctor = (Doctor)event.getTableView().getItems().get(event.getTablePosition().getRow());

System.out.println(doctor);

doctorDao.updateName(doctor,event.getNewValue());

});

MyTableColumn specialityCol = new MyTableColumn("Специальность","speciality",200,true);

specialityCol.tableColumn.setOnEditCommit(event -> {

Doctor doctor = (Doctor)event.getTableView().getItems().get(event.getTablePosition().getRow());

System.out.println(doctor);

doctorDao.updateSpeciality(doctor,event.getNewValue());

});

MyTableColumn birthdayCol = new MyTableColumn("День рождения","birthday",100,true);

birthdayCol.tableColumn.setOnEditCommit(event -> {

Doctor doctor = (Doctor)event.getTableView().getItems().get(event.getTablePosition().getRow());

if (ActionUtils.checkDate(event.getNewValue())!=null)

doctorDao.updateBirthday(doctor,ActionUtils.checkDate(event.getNewValue()));

});

table.getColumns().addAll(idCol.getTc(), nameCol.getTc(), specialityCol.getTc(), birthdayCol.getTc());

if(flagFind||flagFindDaughters)

table.setItems(listDoctors);

else {

table.setItems(FXCollections.observableArrayList(doctorDao.findAll()));

}

break;

}

case "Дети": {

MyTableColumn idCol = new MyTableColumn("Id","id",50,false);

MyTableColumn nameCol = new MyTableColumn("Имя","name",225,true);

nameCol.tableColumn.setOnEditCommit(event -> {

Child child = (Child)event.getTableView().getItems().get(event.getTablePosition().getRow());

childDao.updateName(child,event.getNewValue());

});

MyTableColumn birthdayCol = new MyTableColumn("День рождения","birthday",100,true);

birthdayCol.tableColumn.setOnEditCommit(event -> {

Child child = (Child)event.getTableView().getItems().get(event.getTablePosition().getRow());

if (ActionUtils.checkDate(event.getNewValue())!=null)

childDao.updateBirthday(child,ActionUtils.checkDate(event.getNewValue()));

});

table.getColumns().addAll(idCol.getTc(), nameCol.getTc(), birthdayCol.getTc());

if(flagFind||flagFindDaughters)

table.setItems(listChilds);

else

table.setItems(FXCollections.observableArrayList(childDao.findAll()));

break;

}

case "Матери": {

MyTableColumn idCol = new MyTableColumn("Id","id",50,false);

MyTableColumn nameCol = new MyTableColumn("Имя","name",225,true);

nameCol.tableColumn.setOnEditCommit(event -> {

Mother mother = (Mother)event.getTableView().getItems().get(event.getTablePosition().getRow());

motherDao.updateName(mother,event.getNewValue());

});

MyTableColumn birthdayCol = new MyTableColumn("День рождения","birthday",100,true);

birthdayCol.tableColumn.setOnEditCommit(event -> {

Mother mother = (Mother)event.getTableView().getItems().get(event.getTablePosition().getRow());

if (ActionUtils.checkDate(event.getNewValue())!=null)

motherDao.updateBirthday(mother,ActionUtils.checkDate(event.getNewValue()));

});

table.getColumns().addAll(idCol.getTc(), nameCol.getTc(), birthdayCol.getTc());

if(flagFind||flagFindDaughters)

table.setItems(FXCollections.observableArrayList(listMothers));

else

table.setItems(FXCollections.observableArrayList(motherDao.findAll()));

break;

}

case "Палаты": {

MyTableColumn idCol = new MyTableColumn("Id","id",50,false);

MyTableColumn numberCol = new MyTableColumn("Номер палаты","number",100,true);

numberCol.tableColumn.setOnEditCommit(event -> {

HospitalRoom hospitalRoom = (HospitalRoom) event.getTableView().getItems().get(event.getTablePosition().getRow());

try {

Integer.parseInt(event.getNewValue());

hospitalRoomDao.updateNumber(hospitalRoom, event.getNewValue());

} catch (NumberFormatException exception) {

numberFormat.showAndWait();

}

});

MyTableColumn vipCol = new MyTableColumn("VIP-статус","vip",100,true);

vipCol.tableColumn.setOnEditCommit(event -> {

HospitalRoom hospitalRoom = (HospitalRoom)event.getTableView().getItems().get(event.getTablePosition().getRow());

if (event.getNewValue().equals("+") || (event.getNewValue().equals("-"))) {

hospitalRoomDao.updateVip(hospitalRoom, event.getNewValue());

}

else

vipFormat.showAndWait();

});

MyTableColumn numberOfPlacesCol = new MyTableColumn("Число мест","numberOfPlaces",100,true);

numberOfPlacesCol.tableColumn.setOnEditCommit(event -> {

HospitalRoom hospitalRoom = (HospitalRoom)event.getTableView().getItems().get(event.getTablePosition().getRow());

try {

Integer.parseInt(event.getNewValue());

hospitalRoomDao.updateNumberOfPlace(hospitalRoom, event.getNewValue());

} catch (NumberFormatException exception) {

numberFormat.showAndWait();

}

});

table.getColumns().addAll(idCol.getTc(), numberCol.getTc(), vipCol.getTc(), numberOfPlacesCol.getTc());

if(flagFind||flagFindDaughters)

table.setItems(FXCollections.observableArrayList(listHospitalRoom));

else

table.setItems(FXCollections.observableArrayList(hospitalRoomDao.findAll()));

break;

}

default:

table.getColumns().clear();

break;

}

});

VBox vbox = new VBox(table);

vbox.setPrefHeight(270);

vbox.setPrefWidth(600);

vbox.setLayoutX(0);

vbox.setLayoutY(50);

vbox.setPadding(new Insets(5, 10, 0, 10));

Pane pane = new Pane(lbEnterTable.getLb(),langsComboBox,btnAdd.getBtn(),btnDelete.getBtn(),btnFind.getBtn(),btnReset.getBtn(),

btnDeleteAll.getBtn(),btnChilds.getBtn(),btnDoctors.getBtn(),btnHospitalRoom.getBtn(),btnMother.getBtn(), btnMothers.getBtn());

Pane root = new Pane(pane, vbox);

Scene scene = new Scene(root, 600, 420, Color.TRANSPARENT);

stage.setTitle("База данных родильного дома");

MyStage stg = new MyStage(stage,scene);

stg.show();

}

}

**classEntity.Child.java**

package classEntity;

import javax.persistence.\*;

import java.util.List;

import java.util.Objects;

@Entity

@Table(name = "child")

public class Child {

@Id

@SequenceGenerator( name = "child\_id\_seq", sequenceName = "child\_id\_seq", allocationSize = 1, initialValue = 1 )

@GeneratedValue( strategy = GenerationType.SEQUENCE, generator = "child\_id\_seq")

private Integer id;

private String name;

private String birthday;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "id\_mother")

private Mother mother;

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(name = "doctor\_child",

joinColumns = { @JoinColumn(name = "id\_child") },

inverseJoinColumns = { @JoinColumn(name = "id\_doctor") })

private List<Doctor> doctors;

public List<Doctor> getDoctors() {

return doctors;

}

public void setDoctors(List<Doctor> doctors) {

this.doctors = doctors;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBirthday() {

return birthday;

}

public void setBirthday(String birthday) {

this.birthday = birthday;

}

public Mother getMother() {

return mother;

}

public void setMother(Mother mother) {

this.mother = mother;

}

public Child(String name, String birthday) {

this.name = name;

this.birthday = birthday;

}

public Child(){

}

@Override

public String toString() {

return "Child{" +

"Id = " + id +

", Name = '" + name + '\'' +

", Birthday = " + birthday +

'}';

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Child)) return false;

Child child = (Child) o;

return Objects.equals(getId(), child.getId());

}

@Override

public int hashCode() {

return Objects.hash(id, name, birthday, mother, doctors);

}

}

**classEntity.Doctor.java**

package classEntity;

import javax.persistence.\*;

import java.util.List;

import java.util.Objects;

@Entity

@Embeddable

@Table(name = "doctor")

public class Doctor {

@Id

@SequenceGenerator( name = "doctor\_id\_seq", sequenceName = "doctor\_id\_seq", allocationSize = 1, initialValue = 1 )

@GeneratedValue( strategy = GenerationType.SEQUENCE, generator = "doctor\_id\_seq")

private Integer id;

private String name;

private String speciality;

private String birthday;

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(name = "doctor\_child",

joinColumns = { @JoinColumn(name = "id\_doctor") },

inverseJoinColumns = { @JoinColumn(name = "id\_child") })

private List<Child> childs;

public List<Child> getChilds() {

return childs;

}

public void setChilds(List<Child> childs) {

this.childs = childs;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getSpeciality() {

return speciality;

}

public void setSpeciality(String speciality) {

this.speciality = speciality;

}

public String getBirthday() {

return birthday;

}

public void setBirthday(String birthday) {

this.birthday = birthday;

}

/\*public String getBirthdayStr() {

return DoctorDao.birthdayStr;

}

public void setBirthdayStr(String birthdayStr) {

this.birthday = Utils.newDate(birthdayStr);

}\*/

public Doctor(String name, String speciality, String birthday) {

this.name = name;

this.speciality = speciality;

this.birthday = birthday;

}

public Doctor() {

}

@Override

public String toString() {

return "Doctor{" +

"Id = " + id +

", Name = '" + name + '\'' +

", Speciality = '" + speciality + '\'' +

", Birthday = " + birthday +

'}';

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Doctor)) return false;

Doctor doctor = (Doctor) o;

return Objects.equals(getId(), doctor.getId());

}

@Override

public int hashCode() {

return Objects.hash(id, name, speciality, birthday);

}

}

**classEntity.Mother.java**

package classEntity;

import javax.persistence.\*;

import java.util.List;

import java.util.Objects;

@Entity

@Table(name = "mother")

public class Mother {

@Id

@SequenceGenerator( name = "mother\_id\_seq", sequenceName = "mother\_id\_seq", allocationSize = 1, initialValue = 1 )

@GeneratedValue( strategy = GenerationType.SEQUENCE, generator = "mother\_id\_seq")

private Integer id;

private String name;

private String birthday;

@OneToMany(fetch = FetchType.LAZY/\*, mappedBy = "mother", cascade = CascadeType.ALL, orphanRemoval = true\*/)

@JoinColumn(name = "id\_mother")

private List<Child> childs;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "id\_hospital\_room")

private HospitalRoom hospitalRoom;

public List<Child> getChilds() {

return childs;

}

public void setChilds(List<Child> childs) {

this.childs = childs;

}

public HospitalRoom getHospitalRoom() {

return hospitalRoom;

}

public void setHospitalRoom(HospitalRoom hospitalRoom) {

this.hospitalRoom = hospitalRoom;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBirthday() {

return birthday;

}

public void setBirthday(String birthday) {

this.birthday = birthday;

}

public Mother(String name, String birthday) {

this.name = name;

this.birthday = birthday;

}

public Mother(){

}

@Override

public String toString() {

return "Mother{" +

"Id = " + id +

", Name = '" + name + '\'' +

", Birthday = " + birthday +

'}';

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Mother)) return false;

Mother mother = (Mother) o;

return Objects.equals(getId(), mother.getId());

}

@Override

public int hashCode() {

return Objects.hash(id, name, birthday, childs, hospitalRoom);

}

}

**classEntity.HospitalRoom.java**

package classEntity;

import javax.persistence.\*;

import java.util.List;

import java.util.Objects;

@Entity

@Table(name = "hospital\_room")

public class HospitalRoom {

@Id

@SequenceGenerator( name = "hospital\_room\_id\_seq", sequenceName = "hospital\_room\_id\_seq", allocationSize = 1, initialValue = 1 )

@GeneratedValue( strategy = GenerationType.SEQUENCE, generator = "hospital\_room\_id\_seq")

private int id;

private String number;

private String vip;

@Column(name = "number\_of\_places")

private String numberOfPlaces;

@OneToMany(fetch = FetchType.LAZY)

@JoinColumn(name = "id\_hospital\_room")

private List<Mother> mothers;

public List<Mother> getMothers() {

return mothers;

}

public void setMothers(List<Mother> mothers) {

this.mothers = mothers;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getNumber() {

return number;

}

public void setNumber(String number) {

this.number = number;

}

public String getVip() {

return vip;

}

public void setVip(String vip) {

this.vip = vip;

}

public String getNumberOfPlaces() {

return numberOfPlaces;

}

public void setNumberOfPlaces(String numberOfPlaces) {

this.numberOfPlaces = numberOfPlaces;

}

/\*public void setVipStr(String vipStr) {

this.vipStr = vipStr;

if (vipStr.equals("+"))

this.vip= true;

else

if (vipStr.equals("-"))

this.vip = false;

else

System.out.println("error");

//исключение

}\*/

public HospitalRoom(String number, String vip, String numberOfPlaces) {

this.number = number;

this.numberOfPlaces = numberOfPlaces;

this.vip = vip;

}

public HospitalRoom(){

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof HospitalRoom)) return false;

HospitalRoom hospitalRoom = (HospitalRoom) o;

return Objects.equals(getId(), hospitalRoom.getId());

}

@Override

public int hashCode() {

return Objects.hash(id, number, numberOfPlaces, vip);

}

@Override

public String toString() {

return "HospitalRoom{" +

"Id=" + id +

", Number='" + number + '\'' +

", Number of places=" + numberOfPlaces +

", VIP=" + vip +

'}';

}

}

**dao.BasisDao.java**

package dao;

import classEntity.\*;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.cfg.AnnotationConfiguration;

import java.io.Serializable;

import java.util.List;

public abstract class BasisDao<PK extends Serializable, T> {

protected Session getSession() throws NullPointerException{

return new AnnotationConfiguration().configure().buildSessionFactory().openSession();

}

//поиск по id

public T findByIdExemplar(Integer id, T entity) {

Session session = getSession();

String strEntity = findEntity(entity);

@SuppressWarnings("unchecked")

List<T> list = (List<T>) session.createQuery("SELECT c FROM "+strEntity+" c WHERE c.id="+id.toString()+"").list();

session.close();

if (list.isEmpty()) {

return null;

}

return list.get(0);

}

//поиск по атрибуту

public List<T> findByAttribute(T entity, String name, String attribute) {

Session session = getSession();

String str = findEntity(entity);

System.out.println(str);

@SuppressWarnings("unchecked")

List<T> list = session.createQuery(

"SELECT s FROM "+str+" s where "+attribute+" = '"+name+"'").list();

session.close();

return list;

}

//поиск всех экземпляров таблицы

public List<T> findAllExemplar(T entity) {

Session session = getSession();

String str = findEntity(entity);

@SuppressWarnings("unchecked")

List<T> list = (List<T>) session.createQuery("select s from "+str+" s").list();

session.close();

return list;

}

//добавить экземпляр

public void addExemplar(T entity) {

Session session = getSession();

session.beginTransaction();

session.save(entity);

session.getTransaction().commit();

session.close();

}

//обновить екземпляр

public void updateExemplar(T entity){

Session session = getSession();

session.beginTransaction();

session.update(entity);

session.getTransaction().commit();

session.close();

}

//удалить все экземпляры

public void deleteAllExemplars(T entity) {

String entityString = findEntity(entity);

Session session = getSession();

Transaction tx1 = session.beginTransaction();

@SuppressWarnings("unchecked")

List<T> list = session.createQuery("select s from "+entityString+" s").list();

for(T s : list)

session.delete(s);

tx1.commit();

session.close();

}

//удалить выбранный экземпляр

public void deleteExemplar(T entity) {

findExemplar(entity);

Session session = getSession();

Transaction tx1 = session.beginTransaction();

session.delete(entity);

tx1.commit();

session.close();

}

//поиск названия сущности

public String findEntity(T entity){

Doctor doctor= new Doctor();

Child child = new Child();

Mother mother = new Mother();

HospitalRoom hospitalRoom = new HospitalRoom();

if (doctor.getClass()==entity.getClass())

return "Doctor";

else

if (child.getClass()==entity.getClass())

return "Child";

else

if (mother.getClass()==entity.getClass())

return "Mother";

else

if (hospitalRoom.getClass()==entity.getClass())

return "HospitalRoom";

else

return null;

}

//поиск экземпляра, на предмет наличия

public boolean findExemplar(T entity){

Session session = getSession();

String str = findEntity(entity);

boolean flag=false;

for (T s: (List<T>) session.createQuery("select s from "+ str +" s").list())

if (entity.equals(s)) {

flag = true;

break;

}

session.close();

if (!flag) {

System.out.println("Not found "+str);

return false;

}

else return true;

}

}

**dao.ChildDao.java**

package dao;

import classEntity.Child;

import classEntity.Doctor;

import classEntity.Mother;

import org.hibernate.Session;

import java.util.List;

public class ChildDao extends BasisDao<Integer, Child>{

private final Child child = new Child();

DoctorDao doctorDao;

MotherDao motherDao;

public Child findById(Integer id) {

return findByIdExemplar(id,child);

}

public List<Child> findByName(String name) {

return findByAttribute(child,name,"name");

}

public List<Child> findByBirthday(String date) {

return findByAttribute(child,date,"birthday");

}

public List<Child> findAll() throws NullPointerException {

return findAllExemplar(child);

}

//находит всех Doctors данного экземпляра

public List<Doctor> findDoctors(Child child) {

Session session = getSession();

int id = child.getId();

@SuppressWarnings("unchecked")

List<Doctor> list = (List<Doctor>)session.createQuery("SELECT d " +

"FROM Child c " +

"Join c.doctors d " +

"WHERE c.id="+id).list();

session.close();

return list;

}

//находит Mother данного экземпляра

public Mother findMother(Child child) {

Session session = getSession();

int id = child.getId();

@SuppressWarnings("unchecked")

List<Mother> list = (List<Mother>)session.createQuery("SELECT m " +

"FROM Child c " +

"Join c.mother m " +

"WHERE c.id="+id).list();

session.close();

if (list.isEmpty())

return null;

return list.get(0);

}

public void add(Child child) {

addExemplar(child);

}

public void updateName(Child child, String name) {

child.setName(name);

updateExemplar(child);

}

public void updateBirthday(Child child, String date){

child.setBirthday(date);

updateExemplar(child);

}

public void updateDoctors(Child child, List<Doctor> doctors){

child.setDoctors(doctors);

updateExemplar(child);

}

public void updateMother(Child child, Mother mother){

child.setMother(mother);

updateExemplar(child);

}

public void delete(Child child) {

deleteCascade(child);

}

public void deleteCascade(Child child){

Mother mother = findMother(child);

if ((mother != null) && (motherDao.findChilds(mother).size() == 1))

motherDao.delete(mother);

deleteExemplar(child);

}

public void deleteAll() {

for (Child child : findAll())

deleteCascade(child);

}

}

**dao.DoctorDao.java**

package dao;

import classEntity.Child;

import classEntity.Doctor;

import org.hibernate.Session;

import java.util.List;

public class DoctorDao extends BasisDao<Integer, Doctor>{

private final Doctor doctor = new Doctor();

ChildDao childDao;

public Doctor findById(Integer id) {

return findByIdExemplar(id,doctor);

}

public List<Doctor> findByName(String name) {

return findByAttribute(doctor,name,"name");

}

public List<Doctor> findBySpeciality(String speciality) {

return findByAttribute(doctor,speciality,"speciality");

}

public List<Doctor> findByBirthday(String date) {

return findByAttribute(doctor,date,"birthday");

}

public List<Doctor> findAll() throws NullPointerException {

return findAllExemplar(doctor);

}

//находит всех Childs данного экземпляра

public List<Child> findChilds(Doctor doctor) {

Session session =getSession();

int id = doctor.getId();

@SuppressWarnings("unchecked")

List<Child> list = (List<Child>)session.createQuery("SELECT c " +

"FROM Doctor d " +

"Join d.childs c " +

"WHERE d.id="+id).list();

session.close();

return list;

}

public void add(Doctor doctor) {

addExemplar(doctor);

}

public void updateName(Doctor doctor, String name) {

doctor.setName(name);

updateExemplar(doctor);

}

public void updateSpeciality(Doctor doctor, String speciality){

doctor.setSpeciality(speciality);

updateExemplar(doctor);

}

public void updateBirthday(Doctor doctor, String date){

doctor.setBirthday(date);

updateExemplar(doctor);

}

public void updateChilds(Doctor doctor, List<Child> childs){

doctor.setChilds(childs);

updateExemplar(doctor);

}

public void delete(Doctor doctor) {

deleteExemplar(doctor);

}

public void deleteAll() {

deleteAllExemplars(doctor);

}

}

**dao.MotherDao.java**

package dao;

import classEntity.Child;

import classEntity.HospitalRoom;

import classEntity.Mother;

import org.hibernate.Session;

import java.util.List;

public class MotherDao extends BasisDao<Integer, Mother> {

private final Mother mother = new Mother();

ChildDao childDao;

HospitalRoomDao hospitalRoomDao;

public Mother findById(Integer id) {

return findByIdExemplar(id,mother);

}

public List<Mother> findByName(String name) {

return findByAttribute(mother,name,"name");

}

public List<Mother> findByBirthday(String date) {

return findByAttribute(mother,date,"birthday");

}

public List<Mother> findAll() throws NullPointerException {

return findAllExemplar(mother);

}

//находит всех Childs данного экземпляра

public List<Child> findChilds(Mother mother) {

Session session =getSession();

int id = mother.getId();

@SuppressWarnings("unchecked")

List<Child> list = (List<Child>)session.createQuery("SELECT c " +

"FROM Mother m " +

"Join m.childs c " +

"WHERE m.id="+id).list();

session.close();

return list;

}

//находит HospitalRoom данного экземпляра

public HospitalRoom findHospitalRoom(Mother mother) {

Session session =getSession();

int id = mother.getId();

@SuppressWarnings("unchecked")

List<HospitalRoom> list = (List<HospitalRoom>)session.createQuery("SELECT h " +

"FROM Mother m " +

"Join m.hospitalRoom h " +

"WHERE m.id="+id).list();

session.close();

if (list.isEmpty())

return null;

return list.get(0);

}

public void add(Mother mother) {

addExemplar(mother);

}

public void updateName(Mother mother, String name) {

mother.setName(name);

updateExemplar(mother);

}

public void updateBirthday(Mother mother, String date){

mother.setBirthday(date);

updateExemplar(mother);

}

public void updateChilds(Mother mother, List<Child> childs){

mother.setChilds(childs);

updateExemplar(mother);

}

public void updateHospitalRoom(Mother mother, HospitalRoom hospitalRoom){

mother.setHospitalRoom(hospitalRoom);

updateExemplar(mother);

}

public void delete(Mother mother) {

deleteCascade(mother);

}

public void deleteCascade(Mother mother){

for (Child child : findChilds(mother))

childDao.delete(child);

deleteExemplar(mother);

}

public void deleteAll() {

for (Mother mother : findAll())

deleteCascade(mother);

}

}

**dao.HospitalRoomDao.java**

package dao;

import classEntity.HospitalRoom;

import classEntity.Mother;

import org.hibernate.Session;

import java.util.List;

public class HospitalRoomDao extends BasisDao<Integer, HospitalRoom> {

private final HospitalRoom hospitalRoom = new HospitalRoom();

MotherDao motherDao;

public HospitalRoom findById(Integer id) {

return findByIdExemplar(id,hospitalRoom);

}

public List<HospitalRoom> findByNumber(String number) {

return findByAttribute(hospitalRoom,number,"number");

}

public List<HospitalRoom> findByVip(String vip) {

return findByAttribute(hospitalRoom,vip,"vip");

}

public List<HospitalRoom> findByNumberOfPlace(String numberOfPlace) {

return findByAttribute(hospitalRoom,numberOfPlace,"numberOfPlaces");

}

public List<HospitalRoom> findAll() throws NullPointerException {

return findAllExemplar(hospitalRoom);

}

//находит всех Mothers данного экземпляра

public List<Mother> findMothers(HospitalRoom hospitalRoom) {

Session session = getSession();

int id = hospitalRoom.getId();

@SuppressWarnings("unchecked")

List<Mother> list = (List<Mother>)session.createQuery("SELECT m " +

"FROM HospitalRoom h " +

"Join h.mothers m " +

"WHERE h.id="+id).list();

session.close();

return list;

}

public void add(HospitalRoom hospitalRoom) {

addExemplar(hospitalRoom);

}

public void updateNumber(HospitalRoom hospitalRoom, String number) {

hospitalRoom.setNumber(number);

updateExemplar(hospitalRoom);

}

public void updateVip(HospitalRoom hospitalRoom, String vip){

hospitalRoom.setVip(vip);

updateExemplar(hospitalRoom);

}

public void updateNumberOfPlace(HospitalRoom hospitalRoom, String numberOfPlace){

hospitalRoom.setNumberOfPlaces(numberOfPlace);

updateExemplar(hospitalRoom);

}

public void updateMothers(HospitalRoom hospitalRoom, List<Mother> mothers){

hospitalRoom.setMothers(mothers);

updateExemplar(hospitalRoom);

}

public void delete(HospitalRoom hospitalRoom) {

deleteExemplar(hospitalRoom);

}

public void deleteAll() {

deleteAllExemplars(hospitalRoom);

}

}

**elementsJFX.MyAlert.java**

package elementsJFX;

import javafx.scene.control.Alert;

public class MyAlert {

private final Alert alert;

public MyAlert(Alert.AlertType alertType, String ContentText){

alert = new Alert(alertType);

alert.setTitle("Ошибка");

alert.setHeaderText(null);

alert.setContentText(ContentText);

}

public void showAndWait() {

alert.showAndWait();

}

}

**elementsJFX.MyButton.java**

package elementsJFX;

import javafx.scene.control.Button;

import javafx.scene.text.Font;

import javafx.stage.Stage;

public class MyButton {

public Button btn = new Button();

public MyButton(String str, int sizeFont, int prefWidth, int x, int y){

btn.setText(str); //Установка текста

btn.setFont(new Font("Arial", sizeFont)); //Настройка шрифта

btn.setPrefWidth(prefWidth); //Размер кнопки

btn.setLayoutX(x); //Положение по x

btn.setLayoutY(y); //Положение по y

}

public Button getBtn() {

return btn;

}

public void setOnActionExit(Stage stage) {btn.setOnAction(event -> stage.close());}

public void setDisable(boolean flag) {

btn.setDisable(flag);

}

public void setWork(boolean flag){

btn.setDisable(!flag);

btn.setVisible(flag);

}

}

**elementsJFX.MyLabel.java**

package elementsJFX;

import javafx.scene.control.Label;

import javafx.scene.text.Font;

import javafx.scene.text.FontWeight;

import javafx.scene.text.TextAlignment;

public class MyLabel {

public Label lb = new Label();

public MyLabel(String str, int sizeFont, int x, int y){

lb.setText(str); //Установка текста

lb.setFont(Font.font("Arial", FontWeight.BOLD, sizeFont)); //Настройка шрифта

lb.setTextAlignment(TextAlignment.CENTER); //Выравнивание по центру

lb.setWrapText(true); //Перенос текста на следующую строку

lb.setLayoutX(x); //Положение по x

lb.setLayoutY(y); //Положение по y

}

public Label getLb() {

return lb;

}

public String getText() {

return lb.getText();

}

}

**elementsJFX.MyStage.java**

package elementsJFX;

import javafx.scene.Scene;

import javafx.stage.Modality;

import javafx.stage.Stage;

import javafx.stage.Window;

public class MyStage {

private final Stage stg;

public MyStage(Stage stage, Scene scene){

stg = stage;

stg.setResizable(false); //Отключение растягивания окна

stg.setScene(scene); //Подключение Scene к Stage

}

public MyStage(Stage stage, Scene scene, Window window){

stg = stage;

stg.setResizable(false);

stg.setScene(scene);

stg.initOwner(window);

stg.initModality(Modality.WINDOW\_MODAL); //Привязка дочернего окна к родительскому, без возможности закрытия родительского

}

public void show() {

stg.show();

}

public void showAndWait() {

stg.showAndWait();

}

}

**elementsJFX.MyTableColumn.java**

package elementsJFX;

import javafx.scene.control.TableColumn;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.control.cell.TextFieldTableCell;

public class MyTableColumn {

public TableColumn<Object, String> tableColumn;

public MyTableColumn(String nameColumn, String classColumn, int width, boolean flag){

if (flag) {

tableColumn = new TableColumn<>(nameColumn);

tableColumn.setCellValueFactory(new PropertyValueFactory<>(classColumn));

tableColumn.setCellFactory(TextFieldTableCell.forTableColumn());

tableColumn.setMinWidth(width);

}

else {

tableColumn = new TableColumn<>(nameColumn);

tableColumn.setCellValueFactory(new PropertyValueFactory<>(classColumn));

tableColumn.setMinWidth(width);

}

}

public TableColumn<Object,String> getTc(){

return tableColumn;

}

}

**elementsJFX.MyTextField.java**

package elementsJFX;

import javafx.scene.control.TextField;

import javafx.scene.text.Font;

public class MyTextField {

private TextField tf = new TextField();

public MyTextField(int prefWidth, int sizeFont, int x, int y){

tf.setPrefWidth(prefWidth); //Размер строки ввода

tf.setFont(new Font("Arial", sizeFont)); //Настройка шрифта

tf.setLayoutX(x); //Положение по x

tf.setLayoutY(y); //Положение по y

}

public TextField getTf() {

return tf;

}

public String getText() {

return tf.getText();

}

public void setText(String str) {

this.tf.setText(str);

}

}

**actionJFX.ActionChild.java**

package actionJFX;

import classEntity.Child;

import classEntity.Doctor;

import classEntity.Mother;

import dao.ChildDao;

import dao.DoctorDao;

import dao.MotherDao;

import elementsJFX.\*;

import javafx.collections.ObservableList;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

import javafx.stage.Window;

import java.util.ArrayList;

import java.util.List;

public class ActionChild {

private static DoctorDao doctorDao = new DoctorDao();

private static ChildDao childDao = new ChildDao();

private static MotherDao motherDao = new MotherDao();

private static List<String> filter = new ArrayList<>();

static MyAlert numberFormatAlert = new MyAlert(Alert.AlertType.ERROR,"ID должен содержать только цифры");

static MyAlert failureInfo = new MyAlert(Alert.AlertType.ERROR,"Заполнены не все поля!");

static MyAlert exemplarNotFound = new MyAlert(Alert.AlertType.ERROR,"Экземпляр не найден!");

static MyAlert connectionExists = new MyAlert(Alert.AlertType.ERROR,"Такая связь уже существует");

public static void findWindows(Window window){

Stage stage = new Stage();

stage.setTitle("Поиск");

MyLabel lbWelcome = new MyLabel("Выберите критерии для поиска по таблице", 14, 123, 15);

MyLabel lbId = new MyLabel("По id", 12, 40, 50);

MyLabel lbName = new MyLabel("По имени", 12, 200, 50);

MyLabel lbBirthday = new MyLabel("По дню рождения", 12, 370, 50);

MyTextField tfId = new MyTextField(50, 12, 30, 70);

MyTextField tfName = new MyTextField(220, 12, 115, 70);

MyTextField tfBirthday = new MyTextField(100, 12, 370, 70);

MyButton btnNext = new MyButton("Принять", 12, 450, 30, 110);

btnNext.btn.setOnAction(event -> {

if ((!tfId.getText().isEmpty()) || (!tfName.getText().isEmpty()) || (!tfBirthday.getText().isEmpty())) {

if (!filter.isEmpty())

filter.clear();

try {

if (!tfId.getText().isEmpty())

Integer.parseInt(tfId.getText());

filter.add(tfId.getText());

filter.add(tfName.getText());

if (!tfBirthday.getText().isEmpty()) {

if (ActionUtils.checkDate(tfBirthday.getText()) != null) {

filter.add(ActionUtils.checkDate(tfBirthday.getText()));

stage.close();

}

} else {

filter.add(tfBirthday.getText());

stage.close();

}

} catch (NumberFormatException e) {

numberFormatAlert.showAndWait();

}

} else

stage.close();

});

Pane root = new Pane(lbWelcome.getLb(),lbId.getLb(), lbName.getLb(), lbBirthday.getLb(),

tfId.getTf(),tfName.getTf(), tfBirthday.getTf(), btnNext.getBtn());

Scene scene = new Scene(root, 500, 150, Color.TRANSPARENT);

MyStage stg = new MyStage(stage,scene,window);

stg.showAndWait();

}

public static void addWindows(Window window){

Stage stage = new Stage();

stage.setTitle("Добавление");

MyLabel lbWelcome = new MyLabel("Заполните данные об экземпляре", 14, 48, 15);

MyLabel lbName = new MyLabel("Имя", 12, 97, 50);

MyLabel lbBirthday = new MyLabel("День рождения", 12, 208, 50);

MyTextField tfName = new MyTextField(170, 12, 25, 70);

MyTextField tfBirthday = new MyTextField(80, 12, 220, 70);

MyButton btnNext = new MyButton("Принять", 12, 130, 170, 110);

btnNext.btn.setOnAction(event -> {

if ((!tfName.getText().isEmpty()) && (!tfBirthday.getText().isEmpty())) {

if (!filter.isEmpty())

filter.clear();

if (ActionUtils.checkDate(tfBirthday.getText())!=null) {

filter.add(tfName.getText());

filter.add(ActionUtils.checkDate(tfBirthday.getText()));

stage.close();

}

} else

failureInfo.showAndWait();

});

MyButton btnExit = new MyButton("Назад", 12, 130, 25, 110);

btnExit.setOnActionExit(stage);

Pane root = new Pane(lbWelcome.getLb(), lbName.getLb(), lbBirthday.getLb(),

tfName.getTf(), tfBirthday.getTf(), btnNext.getBtn(), btnExit.getBtn());

Scene scene = new Scene(root, 325, 150, Color.TRANSPARENT);

MyStage stg = new MyStage(stage,scene,window);

stg.showAndWait();

}

public static boolean find(Stage stage, ObservableList<Child> listChilds){

findWindows(stage);

if (!listChilds.isEmpty())

listChilds.clear();

if (!filter.isEmpty()) {

if (!filter.get(0).isEmpty()) {

Child child = childDao.findById(Integer.parseInt(filter.get(0)));

if ((filter.get(1).isEmpty() && filter.get(2).isEmpty()) ||

(filter.get(1).equals(child.getName()) && filter.get(2).equals(child.getBirthday())) ||

(filter.get(1).isEmpty() && filter.get(2).equals(child.getBirthday())) ||

(filter.get(2).isEmpty() && filter.get(1).equals(child.getName())))

listChilds.add(child);

} else if (!filter.get(1).isEmpty()) {

List<Child> childs = childDao.findByName(filter.get(1));

for (Child child : childs)

if (filter.get(2).isEmpty() || filter.get(2).equals(child.getBirthday()))

listChilds.add(child);

} else if (!filter.get(2).isEmpty()) {

List<Child> childs = childDao.findByBirthday(filter.get(2));

listChilds.addAll(childs);

}

filter.clear();

return true;

}

else

return false;

}

public static void add(Stage stage, ObservableList<Child> listChilds, boolean flagFindDaughters, Doctor doctor, Mother mother){

if (flagFindDaughters) {

ActionUtils.enterId(stage, filter);

if (!filter.isEmpty()) {

try {

Child childById = childDao.findById(Integer.parseInt(filter.get(0)));

boolean flag = false;

System.out.println(listChilds);

for (Child child : listChilds) {

if (childById.equals(child)) {

flag = true;

break;

}

}

if (!flag) {

listChilds.add(childById);

if (doctor != null)

doctorDao.updateChilds(doctor, listChilds);

if (mother != null)

motherDao.updateChilds(mother, listChilds);

} else

connectionExists.showAndWait();

filter.clear();

}

catch (NullPointerException exception){

exemplarNotFound.showAndWait();

}

}

}

else {

addWindows(stage);

if (!filter.isEmpty()) {

childDao.add(new Child(filter.get(0), filter.get(1)));

filter.clear();

}

}

}

public static void deleteAll(boolean flagFind, boolean flagFindDaughters, ObservableList<Child> listChilds, Doctor doctor,Mother mother){

if (flagFind) {

for (Child child : listChilds)

childDao.delete(child);

listChilds.clear();

}

else

if (flagFindDaughters) {

listChilds.clear();

if (doctor != null)

doctorDao.updateChilds(doctor, null);

if (mother != null)

motherDao.updateChilds(mother, null);

}

else

childDao.deleteAll();

}

public static void delete(boolean flagFind, boolean flagFindDaughters, ObservableList<Child> listChilds, Child child,Doctor doctor,Mother mother){

if (flagFind) {

childDao.delete(child);

listChilds.remove(child);

} else if (flagFindDaughters) {

listChilds.remove(child);

if (doctor != null)

doctorDao.updateChilds(doctor, listChilds);

if (mother != null)

motherDao.updateChilds(mother, listChilds);

} else {

childDao.delete(child);

}

}

}

**actionJFX.ActionDoctor.java**

package actionJFX;

import classEntity.Child;

import classEntity.Doctor;

import dao.ChildDao;

import dao.DoctorDao;

import elementsJFX.\*;

import javafx.collections.ObservableList;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

import javafx.stage.Window;

import java.util.ArrayList;

import java.util.List;

public class ActionDoctor {

private static DoctorDao doctorDao = new DoctorDao();

private static ChildDao childDao = new ChildDao();

private static List<String> filter = new ArrayList<>();

static MyAlert failureInfo = new MyAlert(Alert.AlertType.ERROR,"Заполнены не все поля!");

static MyAlert numberFormatAlert = new MyAlert(Alert.AlertType.ERROR,"ID должен содержать только цифры");

static MyAlert exemplarNotFound = new MyAlert(Alert.AlertType.ERROR,"Экземпляр не найден!");

static MyAlert connectionExists = new MyAlert(Alert.AlertType.ERROR,"Такая связь уже существует");

public static void findWindows(Window window){

Stage stage = new Stage();

stage.setTitle("Поиск");

MyLabel lbWelcome = new MyLabel("Выберите критерии для поиска по таблице", 14, 60, 15);

MyLabel lbId = new MyLabel("По id", 12, 40, 50);

MyLabel lbName = new MyLabel("По имени", 12, 210, 50);

MyLabel lbSpeciality = new MyLabel("По специальности", 12, 80, 100);

MyLabel lbBirthday = new MyLabel("По дню рождения", 12, 265, 100);

MyTextField tfId = new MyTextField(50, 12, 30, 70);

MyTextField tfName = new MyTextField(250, 12, 115, 70);

MyTextField tfSpeciality = new MyTextField(200, 12, 30, 120);

MyTextField tfBirthday = new MyTextField(100, 12, 265, 120);

MyButton btnNext = new MyButton("Применить", 12, 335, 30, 160);

btnNext.btn.setOnAction(event -> {

if ((!tfId.getText().isEmpty()) || (!tfName.getText().isEmpty()) ||

(!tfSpeciality.getText().isEmpty()) || (!tfBirthday.getText().isEmpty())) {

if (!filter.isEmpty())

filter.clear();

try {

if (!tfId.getText().isEmpty())

Integer.parseInt(tfId.getText());

filter.add(tfId.getText());

filter.add(tfName.getText());

filter.add(tfSpeciality.getText());

if (!tfBirthday.getText().isEmpty()) {

if (ActionUtils.checkDate(tfBirthday.getText()) != null) {

filter.add(ActionUtils.checkDate(tfBirthday.getText()));

stage.close();

}

} else {

filter.add(tfBirthday.getText());

stage.close();

}

} catch (NumberFormatException e) {

numberFormatAlert.showAndWait();

}

} else

stage.close();

});

Pane root = new Pane(lbWelcome.getLb(), lbId.getLb(), lbName.getLb(), lbSpeciality.getLb(), lbBirthday.getLb(),

tfId.getTf(), tfName.getTf(), tfSpeciality.getTf(), tfBirthday.getTf(), btnNext.getBtn());

Scene scene = new Scene(root, 400, 200, Color.TRANSPARENT);

MyStage stg = new MyStage(stage,scene,window);

stg.showAndWait();

}

public static void addWindows(Window window){

Stage stage = new Stage();

stage.setTitle("Добавление");

MyLabel lbWelcome = new MyLabel("Заполните данные об экземпляре", 14, 135, 15);

MyLabel lbName = new MyLabel("Имя", 12, 97, 50);

MyLabel lbSpeciality = new MyLabel("Специальность", 12, 247, 50);

MyLabel lbBirthday = new MyLabel("День рождения", 12, 383, 50);

MyTextField tfName = new MyTextField(170, 12, 25, 70);

MyTextField tfSpeciality = new MyTextField(150, 12, 220, 70);

MyTextField tfBirthday = new MyTextField(80, 12, 395, 70);

MyButton btnNext = new MyButton("Добавить", 12, 200, 275, 110);

btnNext.btn.setOnAction(event -> {

if ((!tfName.getText().isEmpty()) && (!tfSpeciality.getText().isEmpty()) && (!tfBirthday.getText().isEmpty())) {

if (!filter.isEmpty())

filter.clear();

if (ActionUtils.checkDate(tfBirthday.getText())!=null) {

filter.add(tfName.getText());

filter.add(tfSpeciality.getText());

filter.add(ActionUtils.checkDate(tfBirthday.getText()));

stage.close();

}

} else

failureInfo.showAndWait();

});

MyButton btnExit = new MyButton("Назад", 12, 200, 25, 110);

btnExit.setOnActionExit(stage);

Pane root = new Pane(lbWelcome.getLb(), lbName.getLb(), lbSpeciality.getLb(), lbBirthday.getLb(),

tfName.getTf(), tfSpeciality.getTf(), tfBirthday.getTf(), btnNext.getBtn(), btnExit.getBtn());

Scene scene = new Scene(root, 500, 150, Color.TRANSPARENT);

MyStage stg = new MyStage(stage,scene,window);

stg.showAndWait();

}

public static boolean find(Stage stage, ObservableList<Doctor> listDoctors){

findWindows(stage);

if (!listDoctors.isEmpty())

listDoctors.clear();

if (!filter.isEmpty()) {

if (!filter.get(0).isEmpty()) {

Doctor doctor = doctorDao.findById(Integer.parseInt(filter.get(0)));

if ((filter.get(1).isEmpty() && filter.get(2).isEmpty() && filter.get(3).isEmpty()) ||

(filter.get(1).equals(doctor.getName()) && filter.get(2).equals(doctor.getSpeciality()) && filter.get(3).equals(doctor.getBirthday())) ||

(filter.get(1).isEmpty() && filter.get(2).equals(doctor.getSpeciality()) && filter.get(3).equals(doctor.getBirthday())) ||

(filter.get(2).isEmpty() && filter.get(1).equals(doctor.getName()) && filter.get(3).equals(doctor.getBirthday())) ||

(filter.get(3).isEmpty() && filter.get(2).equals(doctor.getSpeciality()) && filter.get(1).equals(doctor.getName())) ||

(filter.get(1).isEmpty() && filter.get(2).isEmpty() && filter.get(3).equals(doctor.getBirthday())) ||

(filter.get(1).isEmpty() && filter.get(3).isEmpty() && filter.get(2).equals(doctor.getSpeciality())) ||

(filter.get(2).isEmpty() && filter.get(3).isEmpty() && filter.get(1).equals(doctor.getName())))

listDoctors.add(doctor);

} else

if (!filter.get(1).isEmpty()) {

List<Doctor> doctors = doctorDao.findByName(filter.get(1));

for (Doctor doctor : doctors) {

if ((filter.get(2).isEmpty() && filter.get(3).isEmpty()) ||

(filter.get(2).equals(doctor.getSpeciality()) && filter.get(3).equals(doctor.getBirthday())) ||

(filter.get(2).isEmpty() && filter.get(3).equals(doctor.getBirthday())) ||

(filter.get(3).isEmpty() && filter.get(2).equals(doctor.getSpeciality())))

listDoctors.add(doctor);

}

} else

if (!filter.get(2).isEmpty()) {

List<Doctor> doctors = doctorDao.findBySpeciality(filter.get(2));

for (Doctor doctor : doctors)

if (filter.get(3).isEmpty() || filter.get(3).equals(doctor.getBirthday()))

listDoctors.add(doctor);

} else

if (!filter.get(3).isEmpty()) {

List<Doctor> doctors = doctorDao.findByBirthday(filter.get(3));

listDoctors.addAll(doctors);

}

filter.clear();

return true;

}

else

return false;

}

public static void add(Stage stage, ObservableList<Doctor> listDoctors, boolean flagFindDaughters, Child child){

if (flagFindDaughters) {

ActionUtils.enterId(stage, filter);

if (!filter.isEmpty()) {

try {

Doctor doctorById = doctorDao.findById(Integer.parseInt(filter.get(0)));

boolean flag = false;

for (Doctor doctor : listDoctors) {

if (doctorById.equals(doctor)) {

flag = true;

break;

}

}

if (!flag) {

listDoctors.add(doctorById);

childDao.updateDoctors(child, listDoctors);

}

else

connectionExists.showAndWait();

filter.clear();

}

catch (NullPointerException exception) {

exemplarNotFound.showAndWait();

}

}

}

else {

addWindows(stage);

if (!filter.isEmpty()) {

doctorDao.add(new Doctor(filter.get(0), filter.get(1),filter.get(2)));

filter.clear();

}

}

}

public static void deleteAll(boolean flagFind, boolean flagFindDaughters, ObservableList<Doctor> listDoctors, Child child){

if (flagFind) {

for (Doctor doctor : listDoctors)

doctorDao.delete(doctor);

listDoctors.clear();

}

else

if (flagFindDaughters) {

childDao.updateDoctors(child, null);

listDoctors.clear();

}

else

doctorDao.deleteAll();

}

public static void delete(boolean flagFind, boolean flagFindDaughters, ObservableList<Doctor> listDoctors, Doctor doctor,Child child){

if (flagFind) {

doctorDao.delete(doctor);

listDoctors.remove(doctor);

} else if (flagFindDaughters) {

listDoctors.remove(doctor);

childDao.updateDoctors(child, listDoctors);

} else {

doctorDao.delete(doctor);

}

}

}

**actionJFX.ActionMother.java**

package actionJFX;

import classEntity.Child;

import classEntity.HospitalRoom;

import classEntity.Mother;

import dao.ChildDao;

import dao.HospitalRoomDao;

import dao.MotherDao;

import elementsJFX.\*;

import javafx.collections.ObservableList;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

import javafx.stage.Window;

import java.util.ArrayList;

import java.util.List;

public class ActionMother {

private static HospitalRoomDao hospitalRoomDao = new HospitalRoomDao();

private static ChildDao childDao = new ChildDao();

private static MotherDao motherDao = new MotherDao();

private static List<String> filter = new ArrayList<>();

static MyAlert numberFormatAlert = new MyAlert(Alert.AlertType.ERROR, "ID должен содержать только цифры");

static MyAlert failureInfo = new MyAlert(Alert.AlertType.ERROR, "Заполнены не все поля!");

static MyAlert exemplarNotFound = new MyAlert(Alert.AlertType.ERROR, "Экземпляр не найден!");

static MyAlert hospitalRoomIsFull = new MyAlert(Alert.AlertType.ERROR, "Палата заполнена!");

static MyAlert connectionExists = new MyAlert(Alert.AlertType.ERROR, "Такая связь уже существует");

public static void findWindows(Window window) {

Stage stage = new Stage();

stage.setTitle("Поиск");

MyLabel lbWelcome = new MyLabel("Выберите критерии для поиска по таблице", 14, 123, 15);

MyLabel lbId = new MyLabel("По id", 12, 40, 50);

MyLabel lbName = new MyLabel("По имени", 12, 200, 50);

MyLabel lbBirthday = new MyLabel("По дню рождения", 12, 370, 50);

MyTextField tfId = new MyTextField(50, 12, 30, 70);

MyTextField tfName = new MyTextField(220, 12, 115, 70);

MyTextField tfBirthday = new MyTextField(100, 12, 370, 70);

MyButton btnNext = new MyButton("Принять", 12, 450, 30, 110);

btnNext.btn.setOnAction(event -> {

if ((!tfId.getText().isEmpty()) || (!tfName.getText().isEmpty()) || (!tfBirthday.getText().isEmpty())) {

if (!filter.isEmpty())

filter.clear();

try {

if (!tfId.getText().isEmpty())

Integer.parseInt(tfId.getText());

filter.add(tfId.getText());

filter.add(tfName.getText());

if (!tfBirthday.getText().isEmpty()) {

if (ActionUtils.checkDate(tfBirthday.getText()) != null) {

filter.add(ActionUtils.checkDate(tfBirthday.getText()));

stage.close();

}

} else {

filter.add(tfBirthday.getText());

stage.close();

}

} catch (NumberFormatException e) {

numberFormatAlert.showAndWait();

}

} else

stage.close();

});

Pane root = new Pane(lbWelcome.getLb(), lbId.getLb(), lbName.getLb(), lbBirthday.getLb(),

tfId.getTf(), tfName.getTf(), tfBirthday.getTf(), btnNext.getBtn());

Scene scene = new Scene(root, 500, 150, Color.TRANSPARENT);

MyStage stg = new MyStage(stage, scene, window);

stg.showAndWait();

}

public static void addWindows(Window window) {

Stage stage = new Stage();

stage.setTitle("Добавление");

MyLabel lbWelcome = new MyLabel("Заполните данные об экземпляре", 14, 48, 15);

MyLabel lbName = new MyLabel("Имя", 12, 97, 50);

MyLabel lbBirthday = new MyLabel("День рождения", 12, 208, 50);

MyTextField tfName = new MyTextField(170, 12, 25, 70);

MyTextField tfBirthday = new MyTextField(80, 12, 220, 70);

MyButton btnNext = new MyButton("Принять", 12, 130, 170, 110);

btnNext.btn.setOnAction(event -> {

if ((!tfName.getText().isEmpty()) && (!tfBirthday.getText().isEmpty())) {

if (!filter.isEmpty())

filter.clear();

if (ActionUtils.checkDate(tfBirthday.getText()) != null) {

filter.add(tfName.getText());

filter.add(ActionUtils.checkDate(tfBirthday.getText()));

stage.close();

}

} else

failureInfo.showAndWait();

});

MyButton btnExit = new MyButton("Назад", 12, 130, 25, 110);

btnExit.setOnActionExit(stage);

Pane root = new Pane(lbWelcome.getLb(), lbName.getLb(), lbBirthday.getLb(),

tfName.getTf(), tfBirthday.getTf(), btnNext.getBtn(), btnExit.getBtn());

Scene scene = new Scene(root, 325, 150, Color.TRANSPARENT);

MyStage stg = new MyStage(stage, scene, window);

stg.showAndWait();

}

public static boolean find(Stage stage, ObservableList<Mother> listMothers) {

findWindows(stage);

if (!listMothers.isEmpty())

listMothers.clear();

if (!filter.isEmpty()) {

if (!filter.get(0).isEmpty()) {

Mother mother = motherDao.findById(Integer.parseInt(filter.get(0)));

if ((filter.get(1).isEmpty() && filter.get(2).isEmpty()) ||

(filter.get(1).equals(mother.getName()) && filter.get(2).equals(mother.getBirthday())) ||

(filter.get(1).isEmpty() && filter.get(2).equals(mother.getBirthday())) ||

(filter.get(2).isEmpty() && filter.get(1).equals(mother.getName())))

listMothers.add(mother);

} else if (!filter.get(1).isEmpty()) {

List<Mother> mothers = motherDao.findByName(filter.get(1));

for (Mother mother : mothers)

if (filter.get(2).isEmpty() || filter.get(2).equals(mother.getBirthday()))

listMothers.add(mother);

} else if (!filter.get(2).isEmpty()) {

List<Mother> mothers = motherDao.findByBirthday(filter.get(2));

listMothers.addAll(mothers);

}

filter.clear();

return true;

} else

return false;

}

public static void add(Stage stage, ObservableList<Mother> listMothers, boolean flagFindDaughters, Child child, HospitalRoom hospitalRoom) {

if (flagFindDaughters) {

ActionUtils.enterId(stage, filter);

if (!filter.isEmpty()) {

try {

Mother motherById = motherDao.findById(Integer.parseInt(filter.get(0)));

boolean flag = false;

for (Mother mother : listMothers) {

if (motherById.equals(mother)) {

flag = true;

break;

}

}

if (!flag) {

if (child != null) {

listMothers.setAll(motherById);

childDao.updateMother(child, listMothers.get(0));

}

if (hospitalRoom != null) {

if (hospitalRoomDao.findMothers(hospitalRoom).size() != Integer.parseInt(hospitalRoom.getNumberOfPlaces())) {

listMothers.add(motherById);

hospitalRoomDao.updateMothers(hospitalRoom, listMothers);

} else

hospitalRoomIsFull.showAndWait();

}

} else

connectionExists.showAndWait();

filter.clear();

} catch (NullPointerException exception) {

exemplarNotFound.showAndWait();

}

}

} else {

addWindows(stage);

if (!filter.isEmpty()) {

motherDao.add(new Mother(filter.get(0), filter.get(1)));

filter.clear();

}

}

}

public static void deleteAll(boolean flagFind, boolean flagFindDaughters, ObservableList<Mother> listMothers, Child child, HospitalRoom hospitalRoom) {

if (flagFind) {

for (Mother mother : listMothers)

motherDao.delete(mother);

listMothers.clear();

} else if (flagFindDaughters) {

listMothers.clear();

if (child != null)

childDao.updateMother(child, null);

if (hospitalRoom != null)

hospitalRoomDao.updateMothers(hospitalRoom, null);

} else

motherDao.deleteAll();

}

public static void delete(boolean flagFind, boolean flagFindDaughters, ObservableList<Mother> listMothers,Mother mother, Child child, HospitalRoom hospitalRoom){

if (flagFind) {

motherDao.delete(mother);

listMothers.remove(mother);

} else if (flagFindDaughters) {

listMothers.remove(mother);

if (child != null)

childDao.updateMother(child, null);

if (hospitalRoom != null)

hospitalRoomDao.updateMothers(hospitalRoom, listMothers);

} else {

motherDao.delete(mother);

}

}

}

**actionJFX.ActionHospitalRoom.java**

package actionJFX;

import classEntity.\*;

import dao.\*;

import elementsJFX.\*;

import javafx.collections.ObservableList;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

import javafx.stage.Window;

import java.util.ArrayList;

import java.util.List;

public class ActionHospitalRoom {

private static HospitalRoomDao hospitalRoomDao = new HospitalRoomDao();

private static MotherDao motherDao = new MotherDao();

private static List<String> filter = new ArrayList<>();

static MyAlert numberFormatAlert = new MyAlert(Alert.AlertType.ERROR,"Поля: 'Номер палаты', 'Число мест в палате' и 'ID'\n должны содержать только цифры");

static MyAlert vipFormatAlert = new MyAlert(Alert.AlertType.ERROR,"Некорректно введен Vip-статус! (+/-)");

static MyAlert exemplarNotFound = new MyAlert(Alert.AlertType.ERROR, "Экземпляр не найден!");

static MyAlert connectionExists = new MyAlert(Alert.AlertType.ERROR, "Такая связь уже существует");

public static void findWindows(Window window){

Stage stage = new Stage();

stage.setTitle("Поиск");

MyLabel lbWelcome = new MyLabel("Выберите критерии для поиска по таблице", 14, 128, 15);

MyLabel lbId = new MyLabel("по id", 12, 40, 50);

MyLabel lbNumber = new MyLabel("по номеру палаты", 12, 100, 50);

MyLabel lbVip = new MyLabel("по VIP-статусу(да/нет)", 12, 220, 50);

MyLabel lbNumberOfPlace = new MyLabel("по числу мест в палате", 12, 375, 50);

MyTextField tfId = new MyTextField(50, 12, 30, 70);

MyTextField tfNumber = new MyTextField(100, 12, 100, 70);

MyTextField tfVip = new MyTextField(135, 12, 220, 70);

MyTextField tfNumberOfPlace = new MyTextField(130, 12, 375, 70);

MyButton btnNext = new MyButton("Принять", 12, 475, 30, 110);

btnNext.btn.setOnAction(event -> {

if ((!tfId.getText().isEmpty()) || (!tfNumber.getText().isEmpty()) ||

(!tfVip.getText().isEmpty()) || (!tfNumberOfPlace.getText().isEmpty())) {

if (!filter.isEmpty())

filter.clear();

if ((!tfVip.getText().isEmpty()) && (!tfVip.getText().equals("+")) && (!tfVip.getText().equals("-")))

vipFormatAlert.showAndWait();

else {

try {

if (!tfId.getText().isEmpty())

Integer.parseInt(tfId.getText());

if (!tfNumber.getText().isEmpty())

Integer.parseInt(tfNumber.getText());

if (!tfNumberOfPlace.getText().isEmpty())

Integer.parseInt(tfNumberOfPlace.getText());

filter.add(tfId.getText());

filter.add(tfNumber.getText());

filter.add(tfVip.getText());

filter.add(tfNumberOfPlace.getText());

stage.close();

} catch (NumberFormatException e) {

numberFormatAlert.showAndWait();

}

}

} else

stage.close();

});

Pane root = new Pane(lbWelcome.getLb(), lbId.getLb(), lbNumber.getLb(), lbVip.getLb(), lbNumberOfPlace.getLb(),

tfId.getTf(), tfNumber.getTf(), tfVip.getTf(), tfNumberOfPlace.getTf(), btnNext.getBtn());

Scene scene = new Scene(root, 535, 150, Color.TRANSPARENT);

MyStage stg = new MyStage(stage,scene,window);

stg.showAndWait();

}

public static void addWindows(Window window){

Stage stage = new Stage();

stage.setTitle("Добавление");

MyLabel lbWelcome = new MyLabel("Заполните данные об экземпляре", 14, 85, 15);

MyLabel lbNumber = new MyLabel("Номер палаты", 12, 26, 50);

MyLabel lbVip = new MyLabel("Vip-статус(+/-)", 12, 140, 50);

MyLabel lbNumberOfPlace = new MyLabel("Число мест в палате", 12, 268, 50);

MyTextField tfNumber = new MyTextField(100, 12, 25, 70);

MyTextField tfVip = new MyTextField(100, 12, 150, 70);

MyTextField tfNumberOfPlace = new MyTextField(100, 12, 275, 70);

MyButton btnNext = new MyButton("Добавить", 12, 150, 225, 110);

btnNext.btn.setOnAction(event -> {

if ((!lbNumber.getText().isEmpty()) && (!lbVip.getText().isEmpty()) && (!lbNumberOfPlace.getText().isEmpty())) {

if ((!tfVip.getText().equals("+")) && (!tfVip.getText().equals("-")))

vipFormatAlert.showAndWait();

else {

if (!filter.isEmpty())

filter.clear();

try {

Integer.parseInt(tfNumber.getText());

Integer.parseInt(tfNumberOfPlace.getText());

filter.add(tfNumber.getText());

filter.add(tfVip.getText());

filter.add(tfNumberOfPlace.getText());

stage.close();

} catch (NumberFormatException e) {

numberFormatAlert.showAndWait();

}

}

}

});

MyButton btnExit = new MyButton("Назад", 12, 150, 25, 110);

btnExit.setOnActionExit(stage);

Pane root = new Pane(lbWelcome.getLb(), lbNumber.getLb(), lbVip.getLb(), lbNumberOfPlace.getLb(),

tfNumber.getTf(), tfVip.getTf(), tfNumberOfPlace.getTf(), btnNext.getBtn(), btnExit.getBtn());

Scene scene = new Scene(root, 400, 150, Color.TRANSPARENT);

MyStage stg = new MyStage(stage,scene,window);

stg.showAndWait();

}

public static boolean find(Stage stage, ObservableList<HospitalRoom> listHospitalRoom) {

findWindows(stage);

if (!listHospitalRoom.isEmpty())

listHospitalRoom.clear();

if (!filter.isEmpty()) {

if (!filter.get(0).isEmpty()) {

HospitalRoom hospitalRoom = hospitalRoomDao.findById(Integer.parseInt(filter.get(0)));

if ((filter.get(1).isEmpty() && filter.get(2).isEmpty() && filter.get(3).isEmpty()) ||

(filter.get(1).equals(hospitalRoom.getNumber()) && filter.get(2).equals(hospitalRoom.getVip()) && filter.get(3).equals(hospitalRoom.getNumberOfPlaces())) ||

(filter.get(1).isEmpty() && filter.get(2).equals(hospitalRoom.getVip()) && filter.get(3).equals(hospitalRoom.getNumberOfPlaces())) ||

(filter.get(2).isEmpty() && filter.get(1).equals(hospitalRoom.getNumber()) && filter.get(3).equals(hospitalRoom.getNumberOfPlaces())) ||

(filter.get(3).isEmpty() && filter.get(2).equals(hospitalRoom.getVip()) && filter.get(1).equals(hospitalRoom.getNumber())) ||

(filter.get(1).isEmpty() && filter.get(2).isEmpty() && filter.get(3).equals(hospitalRoom.getNumberOfPlaces())) ||

(filter.get(1).isEmpty() && filter.get(3).isEmpty() && filter.get(2).equals(hospitalRoom.getVip())) ||

(filter.get(2).isEmpty() && filter.get(3).isEmpty() && filter.get(1).equals(hospitalRoom.getNumber())))

listHospitalRoom.add(hospitalRoom);

} else

if (!filter.get(1).isEmpty()) {

List<HospitalRoom> hospitalRooms = hospitalRoomDao.findByNumber(filter.get(1));

for (HospitalRoom hospitalRoom : hospitalRooms) {

if ((filter.get(2).isEmpty() && filter.get(3).isEmpty()) ||

(filter.get(2).equals(hospitalRoom.getVip()) && filter.get(3).equals(hospitalRoom.getNumberOfPlaces())) ||

(filter.get(2).isEmpty() && filter.get(3).equals(hospitalRoom.getNumberOfPlaces())) ||

(filter.get(3).isEmpty() && filter.get(2).equals(hospitalRoom.getVip())))

listHospitalRoom.add(hospitalRoom);

}

} else

if (!filter.get(2).isEmpty()) {

List<HospitalRoom> hospitalRooms = hospitalRoomDao.findByVip(filter.get(2));

for (HospitalRoom hospitalRoom : hospitalRooms)

if (filter.get(3).isEmpty() || filter.get(3).equals(hospitalRoom.getNumberOfPlaces()))

listHospitalRoom.add(hospitalRoom);

} else

if (!filter.get(3).isEmpty()) {

List<HospitalRoom> hospitalRooms = hospitalRoomDao.findByNumberOfPlace(filter.get(3));

listHospitalRoom.addAll(hospitalRooms);

}

filter.clear();

return true;

}

else

return false;

}

public static void add(Stage stage, ObservableList<HospitalRoom> listHospitalRoom, boolean flagFindDaughters, Mother mother) {

if (flagFindDaughters) {

ActionUtils.enterId(stage, filter);

if (!filter.isEmpty()) {

try {

HospitalRoom hospitalRoomById = hospitalRoomDao.findById(Integer.parseInt(filter.get(0)));

boolean flag = false;

for (HospitalRoom hospitalRoom : listHospitalRoom) {

if (hospitalRoomById.equals(hospitalRoom)) {

flag = true;

break;

}

}

if (!flag) {

if (mother != null) {

listHospitalRoom.setAll(hospitalRoomById);

motherDao.updateHospitalRoom(mother, listHospitalRoom.get(0));

}

} else

connectionExists.showAndWait();

filter.clear();

}

catch (NullPointerException exception) {

exemplarNotFound.showAndWait();

}

}

}

else {

addWindows(stage);

if (!filter.isEmpty()) {

hospitalRoomDao.add(new HospitalRoom(filter.get(0), filter.get(1), filter.get(2)));

filter.clear();

}

}

}

public static void deleteAll(boolean flagFind, boolean flagFindDaughters, ObservableList<HospitalRoom> listHospitalRoom, Mother mother) {

if (flagFind) {

for (HospitalRoom hospitalRoom : listHospitalRoom)

hospitalRoomDao.delete(hospitalRoom);

listHospitalRoom.clear();

}

else

if (flagFindDaughters) {

listHospitalRoom.clear();

if (mother != null)

motherDao.updateHospitalRoom(mother, null);

}

else

hospitalRoomDao.deleteAll();

}

public static void delete(boolean flagFind, boolean flagFindDaughters, ObservableList<HospitalRoom> listHospitalRoom,HospitalRoom hospitalRoom, Mother mother){

if (flagFind) {

hospitalRoomDao.delete(hospitalRoom);

listHospitalRoom.remove(hospitalRoom);

} else if (flagFindDaughters) {

listHospitalRoom.remove(hospitalRoom);

if (mother != null)

motherDao.updateHospitalRoom(mother, null);

} else {

hospitalRoomDao.delete(hospitalRoom);

}

}

}

**actionJFX.ActionUtils.java**

package actionJFX;

import elementsJFX.\*;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

import javafx.stage.Window;

import java.util.Date;

import java.util.List;

public class ActionUtils {

static MyAlert notCorrectDate = new MyAlert(Alert.AlertType.ERROR,"Некорректно введена дата! (dd.mm.yyyy)");

static MyAlert numberFormatForDate = new MyAlert(Alert.AlertType.ERROR,"Дата должна содержать только цифры");

static MyAlert numberFormatForId = new MyAlert(Alert.AlertType.ERROR,"ID должен содержать только цифры");

static MyAlert failureInfo = new MyAlert(Alert.AlertType.ERROR,"Поле не заполнено!");

public static String checkDate(String birthday) {

String[] dateSplit = birthday.split("\\.");

if (dateSplit.length != 3) {

notCorrectDate.showAndWait();

return null;

}

Date date = new Date();

try {

date.setDate(Integer.parseInt(dateSplit[0]));

date.setMonth(Integer.parseInt(dateSplit[1])-1);

date.setYear(Integer.parseInt(dateSplit[2]));

if ((date.getDate()<10)||(date.getMonth()<9)) {

if ((date.getDate()<10)&&(date.getMonth()<9))

return "0" + date.getDate() + ".0" + (date.getMonth() + 1) + "." + date.getYear();

if (date.getMonth() < 9)

return date.getDate() + ".0" + (date.getMonth() + 1) + "." + date.getYear();

return "0" + date.getDate() + "." + (date.getMonth() + 1) + "." + date.getYear();

}

return date.getDate() + "." + (date.getMonth() + 1) + "." + date.getYear();

}

catch (NumberFormatException e) {

numberFormatForDate.showAndWait();

return null;

}

}

public static void enterId(Window window, List<String> filter) {

Stage stage = new Stage();

stage.setTitle("Ввод id экземпляра");

MyLabel lbWelcome = new MyLabel("Введите id экземпляра", 14, 56, 15);

lbWelcome.lb.setPrefWidth(260);

MyLabel lbId = new MyLabel("id", 12, 15, 60);

MyTextField tfId = new MyTextField(50, 12, 35, 55);

MyButton btnNext = new MyButton("Окей", 12, 140, 110, 55);

btnNext.btn.setOnAction(event -> {

if (!tfId.getText().isEmpty()) {

if (!filter.isEmpty())

filter.clear();

try {

Integer.parseInt(tfId.getText());

filter.add(tfId.getText());

stage.close();

}

catch (NumberFormatException e){

numberFormatForId.showAndWait();

}

}

else

failureInfo.showAndWait();

});

MyButton btnExit = new MyButton("Назад", 12, 235, 15, 90);

btnExit.setOnActionExit(stage);

Pane root = new Pane(lbWelcome.getLb(), lbId.getLb(), tfId.getTf(), btnNext.getBtn(), btnExit.getBtn());

Scene scene = new Scene(root, 270, 130, Color.TRANSPARENT);

MyStage stg = new MyStage(stage, scene, window);

stg.showAndWait();

}

}