МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра вычислительной техники

**КУРСОВАЯ РАБОТА**

**ПО ДИСЦИПЛИНЕ**

**«ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ»**

ВАРИАНТ - 29

Тема: Разработка информационной системы "Регистратура поликлиники"

Группа: АВТ-815

Студент: Аверин М.Р.

Преподаватель: Васюткина И.А.

Новосибирск 2020 г.

Введение

Характерной чертой нашего времени являются информационные технологии. В понятие «информационные технологии» входят компьютерные технологии (оборудование и программное обеспечение) и технологии связи для передачи информации. Необходимость в информатизации обусловливается многими факторами. Огромными потоками информации, которые циркулируют в различных сферах деятельности, временем, затрачиваемым на процесс обработки этой информации и сложностями, связанными с хранением информации.

Внедрение информационной системы позволяет решить эти задачи. В результате чего сокращается время на обработку информации, осуществляется хранение больших объемов информации, ускоряется поиск необходимой информации, и выдача ее в удобном для пользователя виде. В основе любой информационной системы лежит база данных. База данных представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Процесс создания программной системы предполагает несколько этапов:

Исследование предметной области, той сферы, куда будет внедряться информационная система, сбор сведений об объектах, фактах, событиях и процессах, которые участвуют в рассматриваемом виде деятельности.

Проектирование базы данных. Осуществляется выбор модели данных: иерархическая, сетевая или реляционная. Наиболее удобной моделью является реляционная модель. В реляционной базе данных информация организована в виде отношений. Проектирование базы данных предполагает определение отношений и связей между ними. Определяются типы данных, ограничения на значения, задаются значения по умолчанию. Осуществляется выбор полей.

Разработка приложения, для работы с базой данных. Приложение осуществляет следующие задачи: добавление информации в базу данных, удаление информации из базы данных, выдача информации в необходимом для пользователя виде, по заданным запросам.

Тема моей курсовой «Разработка информационной системы "Регистратура поликлиники"». Система должна хранить сведения информацию о пациентах, информацию о мед. персонале, информацию о заболеваниях. Также система должна позволить получить информацию по заданным запросам. И осуществлять распределение врачей по участкам и предоставлять информацию о врачах, обладающих той или иной специальностью.

Внедрение такой программной системы позволит ускорить поиск информации о предметах. Позволит быстро получить информацию о враче, его пациентах, должность, участок к которому он прикреплен.

Разработка программной системы осуществлялась в связке Java + MySQL.

1. Описание предметной области

Основная деятельность регистратуры поликлиники – это сбор и хранение информации о врачах и пациентах, обработка информации, назначение рекомендаций по определённым жалобам пациента, выдача расписаний работы врачей.

Информация, хранимая в базе данных информационной системы регистратуры:

* Сведения об участках.
* Информация о врачах.
* Сведения о приемах.
* Информация о пациентах (карточка пациента).
* Расписание работы врачей.
* Администраторы
* Жалобы пациента
* Рекомендации по выздоровлению

Сведения об участках.

Участок – это территория, которая относится к поликлинике. Каждый участок обслуживается несколькими врачами. Один врач может обслуживать один участок.

Информация об участке должна содержать:

* номер участка;
* информацию о врачах, которые обслуживают участки.

Информация о врачах:

* какой врач: участковый, или профильный;
* фамилия;
* имя;
* отчество;
* пол;
* табличный номер;
* дата рождения;
* его участок;
* специальность;
* дата устройства на работу;
* адрес проживания;
* почта и пароль для работы в системе.

Сведения о приемах.

* врач;
* дата приема;
* время приёма;
* пациент;
* жалобы;
* рекомендации.

Информация о пациентах.

Информация о пациенте хранится в его карточке. Карточка имеет номер. В карточке указывается:

* фамилия;
* имя;
* отчество пациента;
* возраст;
* пол;
* домашний адрес;
* номер паспорта;
* дата заведения карточки;
* страховой полис.

В карточку заносится информация о каждом посещении поликлиники пациентом: дата посещения, жалобы, предварительный диагноз, рекомендации врача, имя врача и специальность.

Расписание работы врачей.

В расписании работы врачей указывается:

* дни и часы приема;
* врач;
* номер кабинета:
* специальность.

Регистрация пациентов производится работниками регистратуры (администраторами), которые заполняют карточки пациентов. Информацию о болезнях и посещениях вносят врачи. В случае увольнения врача, вся информация о его расписании становится недоступной. Уволить врача может только администратор. Он удаляет из базы сведения о враче.

1. Проектирование базы данных

Проектирование базы данных для регистратуры поликлиники осуществлялось методом «сущность - связь». Интсрументом проектирования базы данных является MySQL Worckbench. В описании предметной области для базы данных регистратура поликлиники были выделены следующие сущности:

* участок
* врач
* администратор
* пациент
* жалобы и рекомендации
* прием
* расписание

Модель базы данных

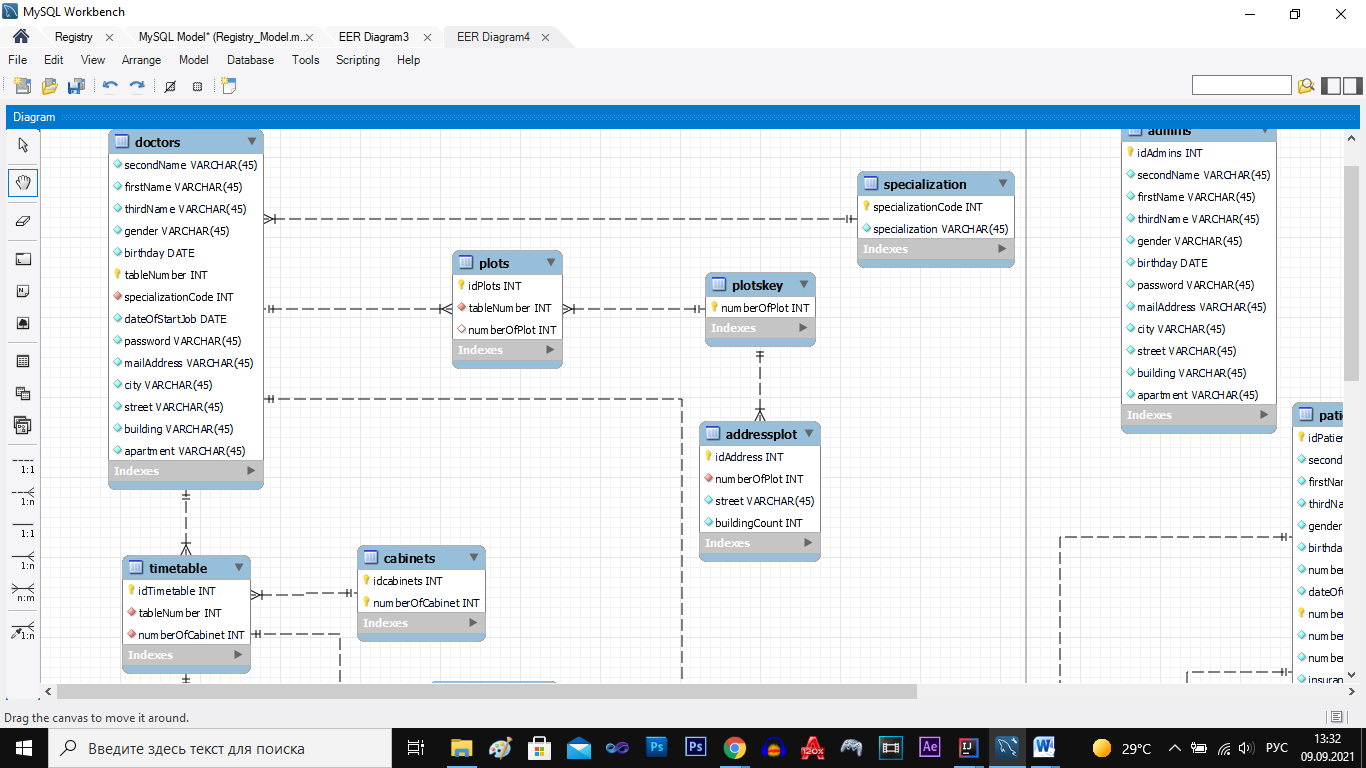


Рис. 1

На Рис.1 изображена модель базы данных для связи врача с участками. Поскольку много врачей могут обслуживать много участков, были разработаны таблицы **plotskey** и **plots** для организации данных в модели, и для предотвращения дублирования записей. Таблица **plotskey** связывает реально существующие адреса Новосибирска с определённым участком врачом в таблице **plots**.

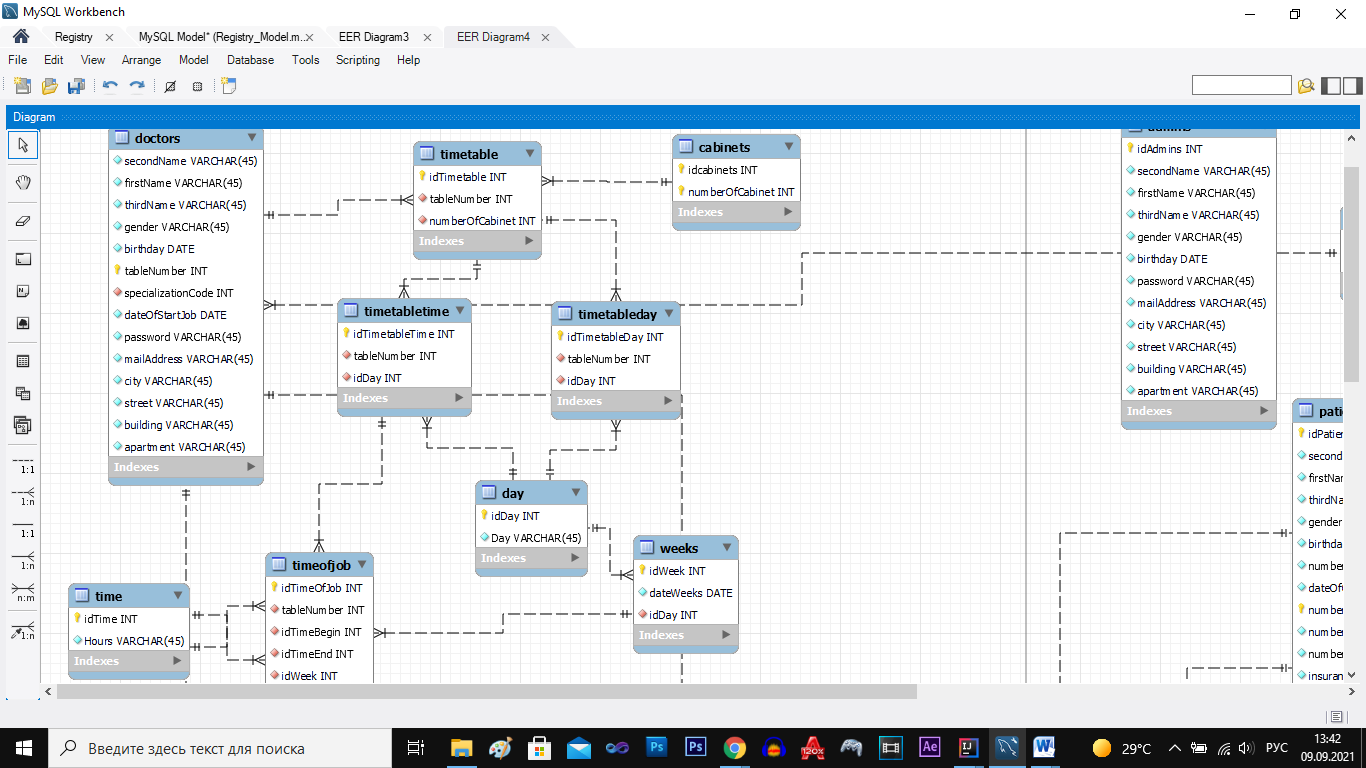


Рис. 2

На Рис.2 изображена модель базы данных для связи врача с расписанием. Поскольку много врачей могут иметь много строк в расписании, что приведёт к дублированию данных, была разработана таблица **timetable,** которая содержит в себе информацию о враче и кабинете, в котором производится приём пациентов, и которая связана соответствующей таблицей, содержащей список доступных кабинетов в поликлинике.

Также **timetable** связана с **timetabletime** и **timetableday,** в которых содержится информация о текущих первых 3 неделях и сами дни недели, соответственно. Эти данные хранятся в равномерно распределённых таблицах для успешной организации хранения:

* **time** содержит список всех доступных часов работы поликлиники с шагом в 30 минут (с 8:00 до 19:00 с учётом обеда)
* **timeofjob** содержит список времен начала и конца рабочего дня для отдельно взятого врача
* **day** содержит список всех доступных дней недели для работы (воскресенье – выходной, остальные дни - рабочие)
* **weeks** содержит в себе текущую неделю и следующие 2; информация по каждому дню является уникальной для каждого врача

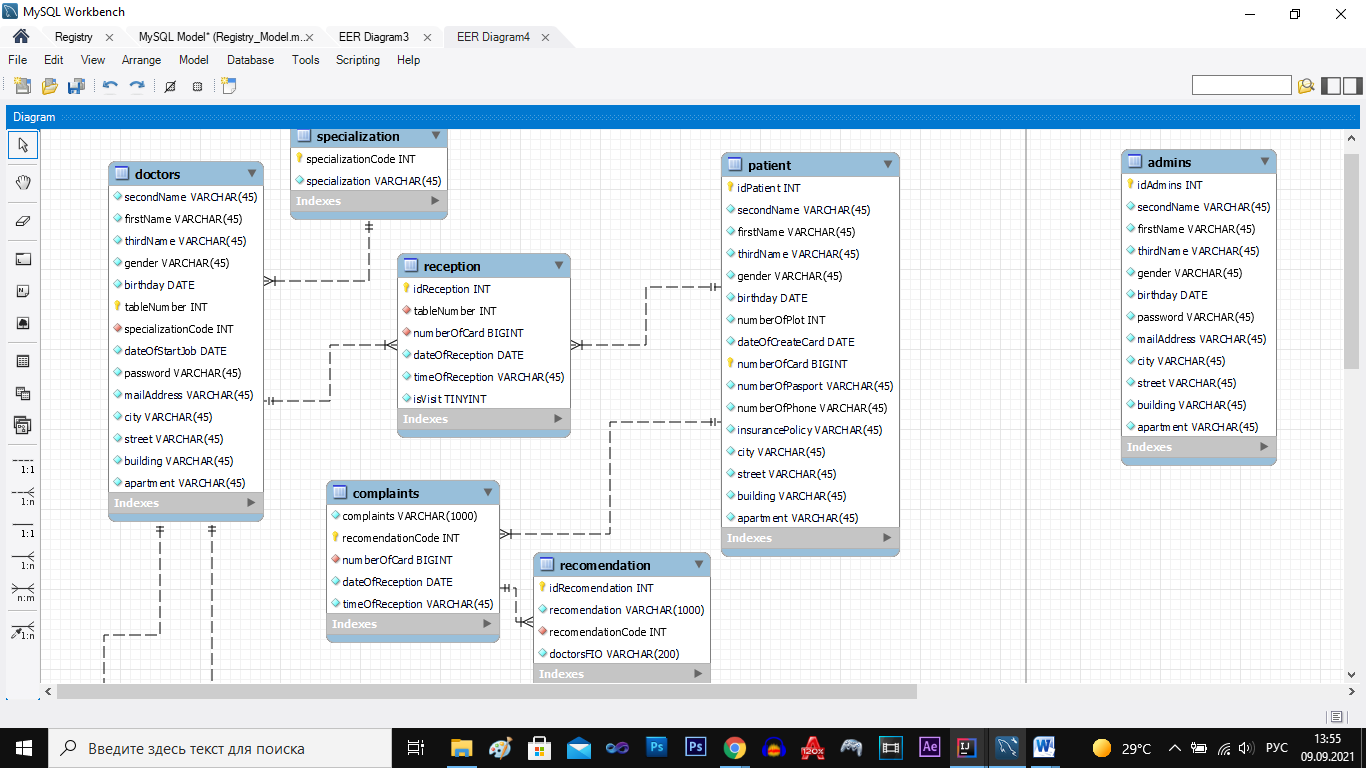


Рис. 3

На Рис.3 изображена модель базы данных для связи врача с его специальностью и пациентом, которого он сможет принять в определённое время и выписать рекомендации по соответствующим жалобам:

* **reception** содержит информацию о запланированном приёме, на который пациенту необходимо прийти в указанное время
* **complaints** содержит информацию о жалобе пациента на определённом приёме
* **recommendation** содержит информацию о рекомендациях, прописанных врачом на соответствующую жалобу, на соответствующем приёме
* **specialization** содержит нумерацию и список доступных специальностей врачей

1. Программа на Java

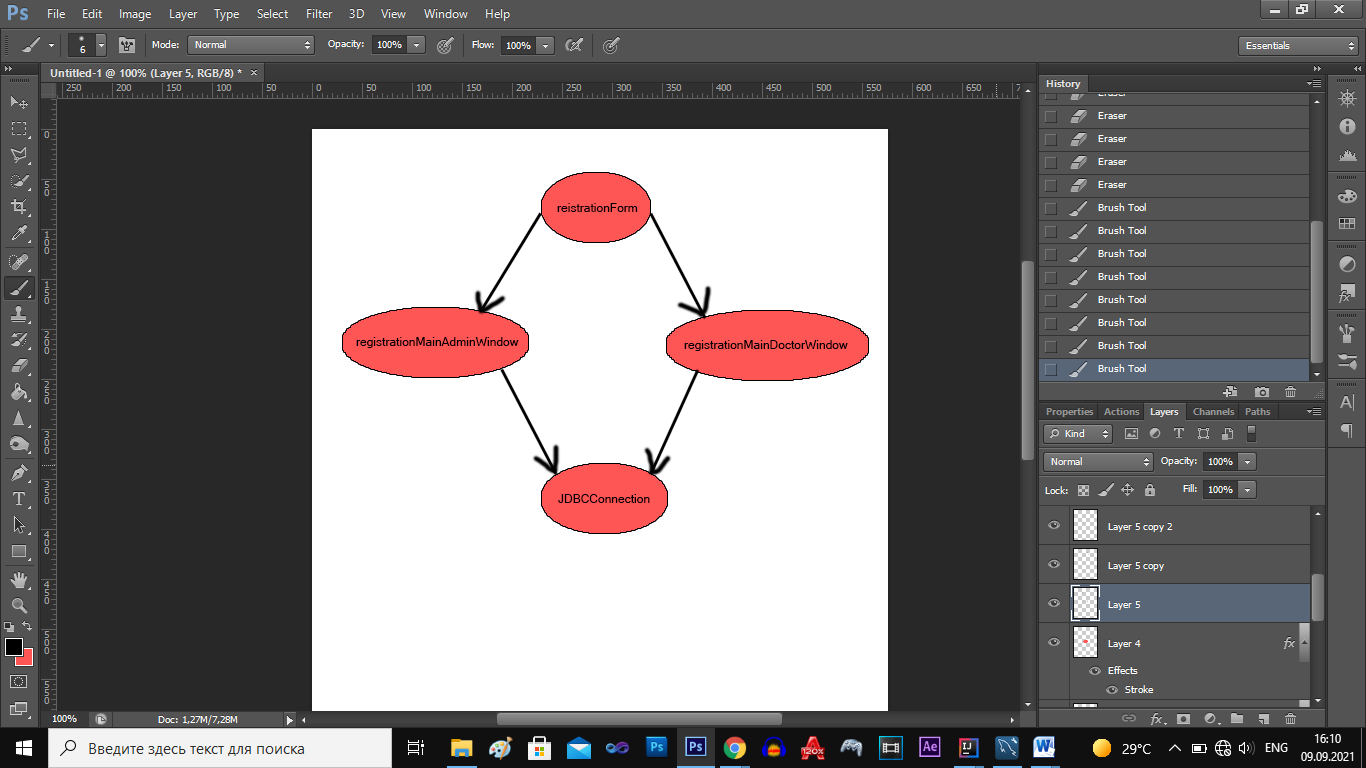
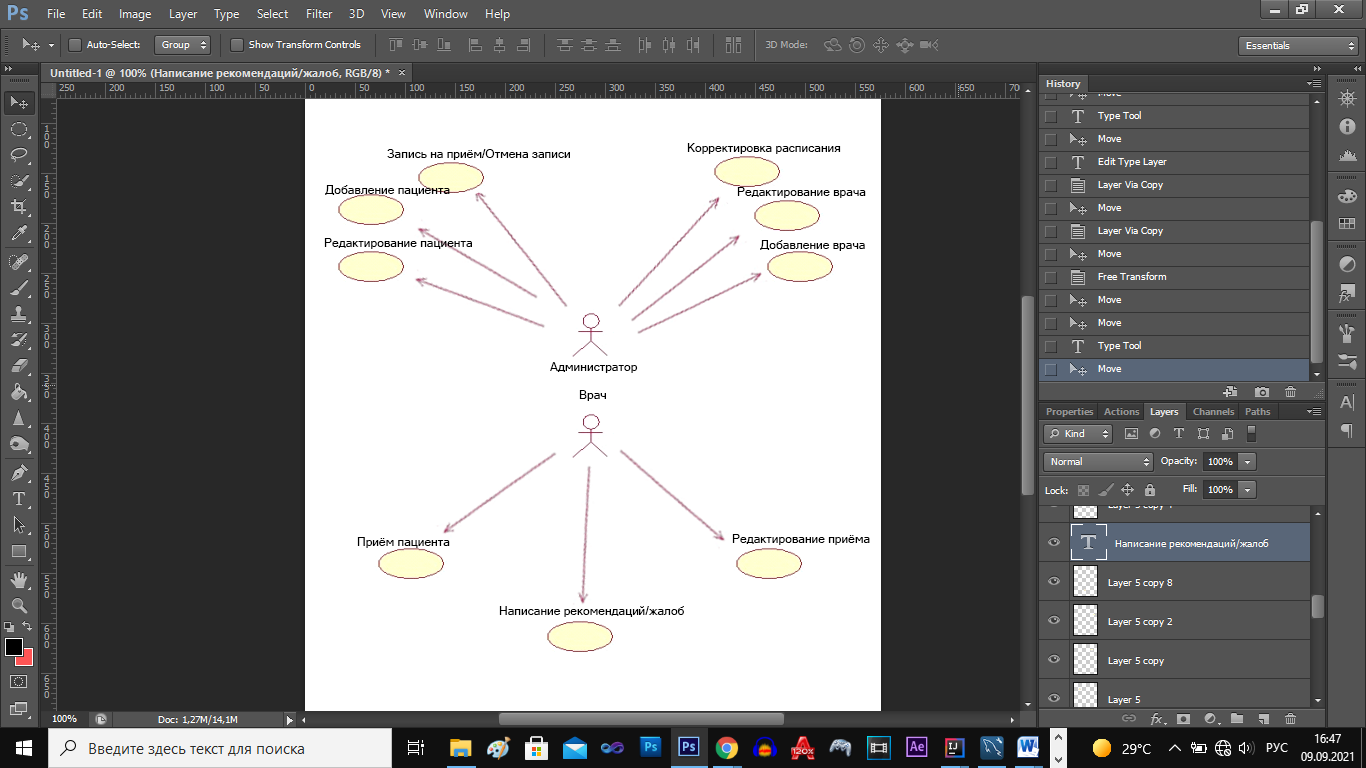
Для эффективного управления данными, была написана программа на Java, с использованием графического интерфейса Swing. Ниже приведена диаграмма классов:

Рис. 4

* registretionForm класс предназначен для организации регистрации администраторов в базе данных и для авторизации врачей и администраторов в системе
* registrationMainAdminWindow класс предназначен для удобной работы администратора с врачами и пациентами. Администратор может редактировать список пациентов и врачей (добавлять, удалять, редактировать и т.д.), расписание врачей и вести учёт пациентов, записанных на приём.
* registrationMainDoctorWindow класс предназначен для удобной работы врача с пациентом. Врач видит список пациентов, записанных к нему на приём, именно к нему.
* JDBCConnection класс предназначен для соединения с базой данных. Все действия со структурой регистратуры осуществляется в нём.

Ниже приведена диаграмма взаимодействия

1. Графический интерфейс
2. Окно регистрации

ПРИЛОЖЕНИЕ

* КЛАСС **registrationForm**
* package registryApplication;  
    
  import Connector.JDBCConnection;  
  import Users.Admin;  
  import Users.Doctor;  
  import com.toedter.calendar.JDateChooser;  
  import net.miginfocom.swing.MigLayout;  
    
  import javax.swing.\*;  
  import java.awt.\*;  
  import java.beans.PropertyChangeEvent;  
  import java.beans.PropertyChangeListener;  
  import java.io.IOException;  
  import java.net.ServerSocket;  
  import java.net.Socket;  
  import java.sql.ResultSet;  
  import java.sql.SQLException;  
  import java.sql.Timestamp;  
  import java.text.SimpleDateFormat;  
  import java.time.LocalDateTime;  
  import java.time.format.DateTimeFormatter;  
  import java.util.\*;  
    
  public class registrationForm extends JFrame implements Runnable  
  {  
  // private String date = "";  
    
   private Dimension screenSize = java.awt.Toolkit.*getDefaultToolkit*().getScreenSize(); // Создаём переменную, в которой хранится размер экрана  
    
   private JPanel cardPanel = new JPanel();  
    
    
   private JPanel startRegistrationForm = new JPanel();  
   private JLabel welcomeLabel = new JLabel("Добро пожаловать!");  
   private JLabel choseYourRollLabel = new JLabel("В качестве кого работать в системе: ");  
   private String doctorString = "Врач", adminString = "Администратор";  
   private String[] rollMassive = {doctorString, adminString};  
   private JComboBox doctor\_adminComboBox = new JComboBox(rollMassive);  
   private JButton registrationButton = new JButton("Регистрация");  
   private JButton enterButton = new JButton("Вход");  
    
    
   private JPanel enterRegistrationFormAdmin = new JPanel();  
   private JPanel enterRegistrationFormDoctor = new JPanel();  
   private JLabel enterDataAdminLabel = new JLabel("Введите данные");  
   private JLabel enterDataDoctorLabel = new JLabel("Введите данные");  
   private JLabel userNameAdminLabel = new JLabel("Логин: ");  
   private JLabel userNameDoctorLabel = new JLabel("Логин: ");  
   private JLabel passwordAdminLabel = new JLabel("Пароль: ");  
   private JLabel passwordDoctorLabel = new JLabel("Пароль: ");  
   private JTextField userNameAdminText = new JTextField();  
   private JTextField userNameDoctorText = new JTextField();  
   private JTextField passwordAdminText = new JTextField();  
   private JTextField passwordDoctorText = new JTextField();  
   private JLabel adminNotDataLabel = new JLabel("Введите все данные!!!");  
   private JLabel doctorNotDataLabel = new JLabel("Введите все данные!!!");  
   private JButton loginAdminButton = new JButton("Войти");  
   private JButton loginDoctorButton = new JButton("Войти");  
    
    
   private JPanel regRegistrationFormAdmin = new JPanel();  
   private JLabel secondNameAdminLabel = new JLabel("Фамилия: ");  
   private JLabel firstNameAdminLabel = new JLabel("Имя: ");  
   private JLabel thirdNameAdminLabel = new JLabel("Отчество: ");  
   private JLabel sexOfAdminLabel = new JLabel("Пол: ");  
   private JLabel maleSexOfAdminLabel = new JLabel("Муж");  
   private JLabel femaleSexOfAdminLabel = new JLabel("Жен");  
   private JLabel birthdayDayAdminLabel = new JLabel("Дата рождения: ");  
   private JLabel newPasswordAdminLabel = new JLabel("Придумайте пароль: ");  
   private JLabel newPasswordAgainAdminLabel = new JLabel("Повторите пароль: ");  
   private JLabel mailAddressAdminLabel = new JLabel("Введите свою почту: ");  
   private JLabel addressOfLiveAdminLabel = new JLabel("Адрес проживания: ");  
   private JLabel cityOfAdminLabel = new JLabel("Город: ");  
   private JLabel streetOfAdminLabel = new JLabel("Улица: ");  
   private JLabel buildingOfAdminLabel = new JLabel("Дом: ");  
   private JLabel apartmentOfAdminLabel = new JLabel("Квартира: ");  
   private JTextField secondNameAdminText = new JTextField();  
   private JTextField firstNameAdminText = new JTextField();  
   private JTextField thirdNameAdminText = new JTextField();  
   private JCheckBox maleAdminCheckBox = new JCheckBox();  
   private JCheckBox femaleAdminCheckBox = new JCheckBox();  
   private JDateChooser birthdayDayAdminTable = new JDateChooser();  
   private JTextField newPasswordAdminText = new JTextField();  
   private JTextField newPasswordAgainAdminText = new JTextField();  
   private JTextField mailAddressAdminText = new JTextField();  
   private JTextField cityOfAdminText = new JTextField();  
   private JTextField streetOfAdminText = new JTextField();  
   private JTextField buildingOfAdminText = new JTextField();  
   private JTextField apartmentOfAdminText = new JTextField();  
   private JLabel notData = new JLabel("Укажите все данные!!!");  
   private JButton registerAdminButton = new JButton("Зарегистрироваться");  
    
   private JButton backEnterAdminButton = new JButton("< Назад");  
   private JButton backEnterDoctorButton = new JButton("< Назад");  
   private JButton backRegAdminButton = new JButton("< Назад");  
    
   private boolean isDoctor = false;  
   private boolean isAdmin = false;  
    
   public registrationForm() // Конструктор класса без параметров  
   {  
  ////////startRegistrationForm(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   startRegistrationForm.setBackground(new Color(255,255,255));  
   startRegistrationForm.setLayout(new MigLayout()); // MigLayout - один из лучших вариантов менеджеров компоновки swing  
    
   doctor\_adminComboBox.setSelectedIndex(1);  
   doctor\_adminComboBox.setFocusable(false);  
    
   startRegistrationForm.add(welcomeLabel,"al center, wrap"); // отступ слева на 110 пикселей, перенос компонентов  
   startRegistrationForm.add(choseYourRollLabel, "gapleft 10, gaptop 50, split 2"); // отступ слева, отступ сверху, объеденение со следующим компонентом  
   startRegistrationForm.add(doctor\_adminComboBox,"wrap push"); // Перенос в самый низ окна  
   startRegistrationForm.add(registrationButton,"al center, split 2");  
   startRegistrationForm.add(enterButton);  
  ////////startRegistrationForm(end)////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
    
  ////////enterRegistrationForm(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   adminNotDataLabel.setForeground(new Color(230,50,50));  
   adminNotDataLabel.setVisible(false);  
   enterRegistrationFormAdmin.setBackground(new Color(255,255,255));  
   enterRegistrationFormAdmin.setLayout(new MigLayout());  
    
   enterRegistrationFormAdmin.add(backEnterAdminButton, "wrap 8");  
   enterRegistrationFormAdmin.add(enterDataAdminLabel, "gapleft 140, wrap");  
   enterRegistrationFormAdmin.add(userNameAdminLabel, "gaptop 15, gapleft 25, split 2");  
   enterRegistrationFormAdmin.add(userNameAdminText, "pushx, growx, wrap");  
   enterRegistrationFormAdmin.add(passwordAdminLabel, "gaptop 10, gapleft 14, split 2");  
   enterRegistrationFormAdmin.add(passwordAdminText, "pushx, growx, wrap");  
   enterRegistrationFormAdmin.add(adminNotDataLabel, "al center, gaptop 20, wrap push");  
   enterRegistrationFormAdmin.add(loginAdminButton, "gaptop 15, gapleft 150");  
    
   doctorNotDataLabel.setForeground(new Color(230,50,50));  
   doctorNotDataLabel.setVisible(false);  
   enterRegistrationFormDoctor.setBackground(new Color(255,255,255));  
   enterRegistrationFormDoctor.setLayout(new MigLayout());  
    
   enterRegistrationFormDoctor.add(backEnterDoctorButton, "wrap 8");  
   enterRegistrationFormDoctor.add(enterDataDoctorLabel, "gapleft 140, wrap");  
   enterRegistrationFormDoctor.add(userNameDoctorLabel, "gaptop 15, gapleft 25, split 2");  
   enterRegistrationFormDoctor.add(userNameDoctorText, "pushx, growx, wrap");  
   enterRegistrationFormDoctor.add(passwordDoctorLabel, "gaptop 10, gapleft 14, split 2");  
   enterRegistrationFormDoctor.add(passwordDoctorText, "pushx, growx, wrap");  
   enterRegistrationFormDoctor.add(doctorNotDataLabel, "al center, gaptop 20, wrap push");  
   enterRegistrationFormDoctor.add(loginDoctorButton, "gaptop 15, gapleft 150");  
  ////////enterRegistrationForm(end)///////////////////////////////////////////////////////////////////////////////////////////////////////////////  
    
  ////////regRegistrationForm(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   regRegistrationFormAdmin.setBackground(new Color(255,255,255));  
   regRegistrationFormAdmin.setLayout(new MigLayout());  
    
   notData.setForeground(new Color(230,50,50));  
   notData.setVisible(false);  
    
   birthdayDayAdminTable.setDateFormatString("yyyy-MM-dd");  
   birthdayDayAdminTable.setDate(new Date());  
    
   regRegistrationFormAdmin.add(backRegAdminButton, "wrap 8");  
   regRegistrationFormAdmin.add(secondNameAdminLabel, "al right");  
   regRegistrationFormAdmin.add(secondNameAdminText, "w 250!, wrap"); // Ограничение длины текстового поля, отступ  
   regRegistrationFormAdmin.add(firstNameAdminLabel, "al right");  
   regRegistrationFormAdmin.add(firstNameAdminText, "w 250!, wrap");  
   regRegistrationFormAdmin.add(thirdNameAdminLabel, "al right");  
   regRegistrationFormAdmin.add(thirdNameAdminText, "w 250!, wrap");  
   regRegistrationFormAdmin.add(sexOfAdminLabel, "al right");  
   regRegistrationFormAdmin.add(maleSexOfAdminLabel, "split 4");  
   regRegistrationFormAdmin.add(maleAdminCheckBox);  
   regRegistrationFormAdmin.add(femaleSexOfAdminLabel, "split 2");  
   regRegistrationFormAdmin.add(femaleAdminCheckBox, "wrap");  
   regRegistrationFormAdmin.add(birthdayDayAdminLabel, "al right");  
   regRegistrationFormAdmin.add(birthdayDayAdminTable, "w 150!, wrap"); //pushx, growx,  
   regRegistrationFormAdmin.add(newPasswordAdminLabel, "al right");  
   regRegistrationFormAdmin.add(newPasswordAdminText, "w 317!, wrap");  
   regRegistrationFormAdmin.add(newPasswordAgainAdminLabel, "al right");  
   regRegistrationFormAdmin.add(newPasswordAgainAdminText, "w 317!, wrap");  
   regRegistrationFormAdmin.add(mailAddressAdminLabel, "al right");  
   regRegistrationFormAdmin.add(mailAddressAdminText, "w 317!, wrap");  
   regRegistrationFormAdmin.add(addressOfLiveAdminLabel, "span 1 4, top"); // Растянуть компонент 1х4  
   regRegistrationFormAdmin.add(cityOfAdminLabel, "gapleft 18, split 2");  
   regRegistrationFormAdmin.add(cityOfAdminText, "w 250!, wrap");  
   regRegistrationFormAdmin.add(streetOfAdminLabel, "gapleft 18, split 2");  
   regRegistrationFormAdmin.add(streetOfAdminText, "w 250!, wrap");  
   regRegistrationFormAdmin.add(buildingOfAdminLabel, "gapleft 30, split 2");  
   regRegistrationFormAdmin.add(buildingOfAdminText, "w 250!, wrap");  
   regRegistrationFormAdmin.add(apartmentOfAdminLabel, "split 2");  
   regRegistrationFormAdmin.add(apartmentOfAdminText, "w 250!, wrap");  
   regRegistrationFormAdmin.add(notData, "span 2 1, al center, gaptop 20, wrap push");  
   regRegistrationFormAdmin.add(registerAdminButton, "span 2 1, al center");  
  ////////regRegistrationForm(end)///////////////////////////////////////////////////////////////////////////////////////////////////////////////  
    
   CardLayout cl = new CardLayout();  
   cardPanel.setLayout(cl); // Менеджер компоновки для переключения между панелями  
   cardPanel.add(startRegistrationForm, "start"); // добавляю на общую карточную панель одну из рабочих панелей  
   cardPanel.add(enterRegistrationFormAdmin, "enterAdmin");  
   cardPanel.add(enterRegistrationFormDoctor, "enterDoctor");  
   cardPanel.add(regRegistrationFormAdmin, "regAdmin");  
   cl.show(cardPanel, "start"); // Выбор панели для первичного отображения  
    
   addRegistrationForm(cardPanel, 390,200);  
    
  ////////actionListeners(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   enterButton.addActionListener(actionEvent -> {  
   if(doctor\_adminComboBox.getSelectedItem().equals(doctorString)) // Форма входа для врача  
   {  
   cl.show(cardPanel, "enterDoctor");  
   refreshRegistrationForm(400,275);  
   }  
   else // Форма входа для администратора  
   {  
   cl.show(cardPanel, "enterAdmin");  
   refreshRegistrationForm(400,275);  
   }  
   });  
   registrationButton.addActionListener(actionEvent -> {  
   if(doctor\_adminComboBox.getSelectedItem().equals(adminString)) // Форма регистрации для врача  
   {  
   cl.show(cardPanel, "regAdmin");  
   refreshRegistrationForm(480,500);  
   }  
   });  
    
   doctor\_adminComboBox.addItemListener(actionEvent -> {  
   if(actionEvent.getItem().equals(doctorString))  
   {  
   registrationButton.setEnabled(false);  
   }  
   else  
   {  
   registrationButton.setEnabled(true);  
   }  
   });  
    
   maleAdminCheckBox.addActionListener(acationEvent -> {  
   if(maleAdminCheckBox.isSelected())  
   {  
   femaleAdminCheckBox.setSelected(false);  
   }  
   });  
   femaleAdminCheckBox.addActionListener(acationEvent -> {  
   if(femaleAdminCheckBox.isSelected())  
   {  
   maleAdminCheckBox.setSelected(false);  
   }  
   });  
    
   backEnterAdminButton.addActionListener(actionEvent -> {  
   cl.show(cardPanel, "start");  
   userNameAdminText.setText(""); // Очистка текстовых полей, заполненных, но не зарегистрированных ранее  
   passwordAdminText.setText("");  
   adminNotDataLabel.setVisible(false);  
   refreshRegistrationForm(390,200);  
   });  
   backEnterDoctorButton.addActionListener(actionEvent -> {  
   cl.show(cardPanel, "start");  
   userNameDoctorText.setText(""); // Очистка текстовых полей, заполненных, но не зарегистрированных ранее  
   passwordDoctorText.setText("");  
   doctorNotDataLabel.setVisible(false);  
   refreshRegistrationForm(390,200);  
   });  
   backRegAdminButton.addActionListener(actionEvent -> {  
   cl.show(cardPanel, "start");  
   birthdayDayAdminTable.setDate(new Date()); // Сброс выбранной ранее даты  
   secondNameAdminText.setText(""); // Очистка текстовых полей, заполненных, но не зарегистрированных ранее  
   firstNameAdminText.setText("");  
   thirdNameAdminText.setText("");  
   newPasswordAdminText.setText("");  
   newPasswordAgainAdminText.setText("");  
   mailAddressAdminText.setText("");  
   cityOfAdminText.setText("");  
   streetOfAdminText.setText("");  
   buildingOfAdminText.setText("");  
   apartmentOfAdminText.setText("");  
   maleAdminCheckBox.setSelected(false);  
   femaleAdminCheckBox.setSelected(false);  
   notData.setVisible(false);  
   refreshRegistrationForm(390,200);  
   });  
    
   registerAdminButton.addActionListener(actionEvent -> {  
   registerNewAdmin();  
   });  
    
   loginAdminButton.addActionListener(actionEvent -> {  
   loginAdmin();  
   });  
   loginDoctorButton.addActionListener(actionEvent -> {  
   loginDoctor();  
   });  
  ////////actionListeners(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   }  
    
   private void loginDoctor()  
   {  
   if(!userNameDoctorText.getText().equals("") && !passwordDoctorText.getText().equals(""))  
   {  
   isDoctor = true;  
   doctorNotDataLabel.setVisible(false);  
    
   JDBCConnection jdbcConnection = new JDBCConnection(); // Создаём бъект класса для работы с БД  
    
   Doctor doctor = new Doctor(); // Вписываем в объект класса необходимые данные (Объект сразу удалится после выполнения метода)  
   doctor.setMailAddress(userNameDoctorText.getText().trim()); // Считал введенные данные пользователем  
   doctor.setPassword(passwordDoctorText.getText()); // Считал введенные данные пользователем  
    
   doctor = jdbcConnection.getDoctor(doctor); // Получаем массив данных обо всех администраторах  
    
   if(!(doctor.getSecondName() == null)) // Вход в систему в качестве администратора  
   {  
   doctorNotDataLabel.setVisible(false);  
    
   this.dispose(); // Закрытие окна регистрации  
   new registryMainDoctorWindow(doctor); // Создание основного рабочего пространства для текущего админа  
   }  
   else // Не найден администратор в БД  
   {  
   doctorNotDataLabel.setText("Врач не найден!!!");  
   doctorNotDataLabel.setVisible(true);  
   }  
   }  
   else  
   {  
   adminNotDataLabel.setText("Введите все данные!!!");  
   doctorNotDataLabel.setVisible(true);  
   }  
   }  
    
   private void loginAdmin()  
   {  
   if(!userNameAdminText.getText().equals("") && !passwordAdminText.getText().equals("")) // Если регистрационные поля не пусты  
   {  
   isAdmin = true;  
   adminNotDataLabel.setVisible(false);  
    
   JDBCConnection jdbcConnection = new JDBCConnection(); // Создаём бъект класса для работы с БД  
    
   Admin admin = new Admin(); // Вписываем в объект класса необходимые данные (Объект сразу удалится после выполнения метода)  
   admin.setMailAddress(userNameAdminText.getText().trim()); // Считал введенные данные пользователем  
   admin.setPassword(passwordAdminText.getText()); // Считал введенные данные пользователем  
    
   admin = jdbcConnection.getAdmin(admin); // Получаем массив данных обо всех администраторах  
    
   if(!(admin.getSecondName() == null)) // Вход в систему в качестве администратора  
   {  
   adminNotDataLabel.setVisible(false);  
    
   this.dispose(); // Закрытие окна регистрации  
   new registryMainAdminWindow(admin); // Создание основного рабочего пространства для текущего админа  
   }  
   else // Не найден администратор в БД  
   {  
   adminNotDataLabel.setText("Администратор не найден!!!");  
   adminNotDataLabel.setVisible(true);  
   }  
   }  
   else // Не все регистрационные поля введены  
   {  
   adminNotDataLabel.setText("Введите все данные!!!");  
   adminNotDataLabel.setVisible(true);  
   }  
   }  
    
   private void registerNewAdmin()  
   {  
   if(!secondNameAdminText.getText().equals("") && !firstNameAdminText.getText().equals("") &&  
   !thirdNameAdminText.getText().equals("") && (maleAdminCheckBox.isSelected() || femaleAdminCheckBox.isSelected()) &&  
   (newPasswordAdminText.getText().equals(newPasswordAgainAdminText.getText()) && !newPasswordAdminText.getText().equals("")) &&  
   !mailAddressAdminText.getText().equals("") && !cityOfAdminText.getText().equals("") && !streetOfAdminText.getText().equals("") &&  
   !buildingOfAdminText.getText().equals("") && !apartmentOfAdminText.getText().equals("") &&  
   birthdayDayAdminTable.getDate().before(new Date())) // Если все регистрационные поля заполнены  
   {  
   notData.setVisible(false);  
    
   JDBCConnection jdbcConnection = new JDBCConnection(); // Создание экземпляра класса для работы с БД  
    
   java.util.Date d = birthdayDayAdminTable.getDate();  
   java.sql.Date d2 = new java.sql.Date(d.getTime());  
    
   String male\_feemale = "";  
   if (maleAdminCheckBox.isSelected()) male\_feemale = "Мужской";  
   else male\_feemale = "Женский";  
    
   Admin admin = new Admin(secondNameAdminText.getText().trim(), firstNameAdminText.getText().trim(),  
   thirdNameAdminText.getText().trim(), male\_feemale, d2,  
   newPasswordAdminText.getText(), mailAddressAdminText.getText().trim(),  
   cityOfAdminText.getText().trim(), streetOfAdminText.getText().trim(),  
   buildingOfAdminText.getText().trim(), apartmentOfAdminText.getText().trim());  
    
   jdbcConnection.registerAdmin(admin);  
    
   birthdayDayAdminTable.setDate(new Date()); // Сброс выбранной ранее даты  
   secondNameAdminText.setText(""); // Очистка текстовых полей, заполненных, но не зарегистрированных ранее  
   firstNameAdminText.setText("");  
   thirdNameAdminText.setText("");  
   newPasswordAdminText.setText("");  
   newPasswordAgainAdminText.setText("");  
   mailAddressAdminText.setText("");  
   cityOfAdminText.setText("");  
   streetOfAdminText.setText("");  
   buildingOfAdminText.setText("");  
   apartmentOfAdminText.setText("");  
   maleAdminCheckBox.setSelected(false);  
   femaleAdminCheckBox.setSelected(false);  
   notData.setVisible(false);  
    
   new infoMessage(this, "Новый администратор зарегистрирован в системе", "УВЕДОМЛЕНИЕ");  
   }  
   else  
   {  
   notData.setVisible(true);  
   }  
   }  
    
   private void addRegistrationForm(JPanel \_carPanel, int width, int height)  
   {  
   this.setBounds(0, 0, width, height); // Устанавливаем оптимальные размеры окна, учитывая размеры экрана  
   this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*); // При нажатии на X произойдёт выход из приложения  
   this.setFocusable(true); // Фокус на окне  
   this.setVisible(true); // Делаем окно видимым  
   this.setResizable(false); // Запрет на изменение размеров окна  
   this.setTitle("Форма регистрации"); // Заголовок окна  
   this.setLocationRelativeTo(null); // Расположение окна ровно по центру  
    
   this.add(\_carPanel);  
   }  
    
   public boolean isAdmin()  
   {  
   return this.isAdmin;  
   }  
    
   public boolean isDoctor()  
   {  
   return this.isDoctor;  
   }  
    
   private void refreshRegistrationForm(int width, int height)  
   {  
   this.setBounds(0, 0, width, height); // Устанавливаем оптимальные размеры окна, учитывая размеры экрана  
   this.setLocationRelativeTo(null); // Расположение окна ровно по центру  
   }  
    
   @Override  
   public void run()  
   {  
    
   }  
  }
* КЛАСС registryMainAdminWindow
* package registryApplication;  
    
  import Connector.JDBCConnection;  
  import Users.Admin;  
  import Users.Doctor;  
  import net.miginfocom.swing.MigLayout;  
  import registryApplication.tableModels.doctorsTableModel;  
  import registryApplication.tableModels.receptionDoneToDoctorTableModel;  
  import registryApplication.tableModels.receptionToDoctorTableModel;  
    
  import javax.swing.\*;  
  import javax.swing.table.AbstractTableModel;  
  import javax.swing.table.TableRowSorter;  
  import java.awt.\*;  
  import java.awt.event.WindowAdapter;  
  import java.awt.event.WindowEvent;  
  import java.io.IOException;  
  import java.io.ObjectInputStream;  
  import java.io.ObjectOutputStream;  
  import java.net.Socket;  
  import java.sql.SQLException;  
  import java.util.ArrayList;  
    
  public class registryMainDoctorWindow extends JFrame  
  {  
   private JDBCConnection jdbcConnection = new JDBCConnection();  
   private Dimension screenSize = java.awt.Toolkit.*getDefaultToolkit*().getScreenSize(); // Создаём переменную, в которой хранится размер экрана  
    
   private JPanel mainDoctorWindowPanel = new JPanel(); // Основная панель  
   private JScrollPane scrollMainDoctorWindowPanel = new JScrollPane(mainDoctorWindowPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
    
   private JPanel doctorTableLeftPanel = new JPanel(); // Диалоговая панель  
    
   private JPanel doctorTableRightPanel = new JPanel(); // Рабочая панель  
    
  ////////////////////////Диалоговая панель(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   private JPanel startRegistryMainDoctorWindow = new JPanel();  
   private JLabel startDoctorLabel = new JLabel("Врач: ");  
   private JLabel startDoctorLabelText = new JLabel();  
   private JLabel startDoctorTableNumberLabel = new JLabel("Табличный номер: ");  
   private JLabel startDoctorTableNumberLabelText = new JLabel();  
   private JLabel startDoctorSpecializationLabel = new JLabel("Специальность: ");  
   private JLabel startDoctorSpecializationLabelText = new JLabel();  
    
   private JButton startDoctorReceptionButton = new JButton("Приёмная");  
   private JButton startDoctorPatientButton = new JButton("Пациенты");  
    
   private JPanel receptionDoctorWindow = new JPanel();  
   private JLabel receptionStartDoctorLabel = new JLabel("Врач: ");  
   private JLabel receptionStartDoctorLabelText = new JLabel();  
   private JLabel receptionStartDoctorTableNumberLabel = new JLabel("Табличный номер: ");  
   private JLabel receptionStartDoctorTableNumberLabelText = new JLabel();  
   private JLabel receptionStartDoctorSpecializationLabel = new JLabel("Специальность: ");  
   private JLabel receptionStartDoctorSpecializationLabelText = new JLabel();  
   private JButton receptionBackToReceptionButton = new JButton("< Назад");  
    
   private JPanel receptionDoctorWindow2 = new JPanel();  
   private JLabel receptionStartDoctorLabel2 = new JLabel("Врач: ");  
   private JLabel receptionStartDoctorLabelText2 = new JLabel();  
   private JLabel receptionStartDoctorTableNumberLabel2 = new JLabel("Табличный номер: ");  
   private JLabel receptionStartDoctorTableNumberLabelText2 = new JLabel();  
   private JLabel receptionStartDoctorSpecializationLabel2 = new JLabel("Специальность: ");  
   private JLabel receptionStartDoctorSpecializationLabelText2 = new JLabel();  
   private JButton receptionBackToReceptionButton2 = new JButton("< Назад");  
  ////////////////////////Диалоговая панель(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
    
  ////////////////////////Рабочая панель(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   private JPanel voidPanel = new JPanel();  
    
   private JPanel propertiesAndListOfReceptionPanel = new JPanel();  
   private JScrollPane scrollPropertyAndListOfReceptionPanel = new JScrollPane(propertiesAndListOfReceptionPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
    
   private JPanel listOfReceptionPanel = new JPanel();  
   private JTable tableOfReceptionToDoctor = new JTable();  
   private receptionToDoctorTableModel rtdtm = new receptionToDoctorTableModel(tableOfReceptionToDoctor);  
   private TableRowSorter<AbstractTableModel> rowSorter = new TableRowSorter(rtdtm);  
   private JScrollPane scrollTableOfReceptionToDoctor = new JScrollPane(tableOfReceptionToDoctor, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
    
   private JPanel propertiesListOfReceptionPanel = new JPanel();  
   private JButton protectListOfReceptionButton = new JButton("Принять");  
    
    
   private JPanel propertiesAndListOfReceptionPanel2 = new JPanel();  
   private JScrollPane scrollPropertyAndListOfReceptionPanel2 = new JScrollPane(propertiesAndListOfReceptionPanel2, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
    
   private JPanel listOfReceptionPanel2 = new JPanel();  
   private JTable tableOfReceptionToDoctor2 = new JTable();  
   private receptionDoneToDoctorTableModel rtdtm2 = new receptionDoneToDoctorTableModel(tableOfReceptionToDoctor2);  
   private TableRowSorter<AbstractTableModel> rowSorter2 = new TableRowSorter(rtdtm2);  
   private JScrollPane scrollTableOfReceptionToDoctor2 = new JScrollPane(tableOfReceptionToDoctor2, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
    
   private JPanel propertiesListOfReceptionPanel2 = new JPanel();  
   private JButton editListOfReceptionButton2 = new JButton("Редактировать");  
  ////////////////////////Рабочая панель(end)///////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   private Doctor doctor;  
    
   public registryMainDoctorWindow(Doctor doctor) // Конструктор класса без параметров  
   {  
   this.doctor = doctor;  
   voidPanel.setBackground(new Color(100,100,100));  
  /////////////////Рабочая панель(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   startRegistryMainDoctorWindow.setBackground(new Color(255,255,255));  
   startRegistryMainDoctorWindow.setLayout(new MigLayout());  
   startDoctorLabelText.setText(doctor.getSecondName() + " " + doctor.getFirstName() + " " + doctor.getThirdName());  
   startDoctorTableNumberLabelText.setText(String.*valueOf*(doctor.getTableNumber()));  
   startDoctorSpecializationLabelText.setText(doctor.getSpecialization());  
    
   startRegistryMainDoctorWindow.add(startDoctorLabel, "wrap 5, al left");  
   startRegistryMainDoctorWindow.add(startDoctorLabelText, "wrap 10, al left");  
   startRegistryMainDoctorWindow.add(startDoctorTableNumberLabel, "split 2, al left");  
   startRegistryMainDoctorWindow.add(startDoctorTableNumberLabelText, "wrap 10");  
   startRegistryMainDoctorWindow.add(startDoctorSpecializationLabel, "wrap 5, al left");  
   startRegistryMainDoctorWindow.add(startDoctorSpecializationLabelText, "wrap 50, al left");  
   startRegistryMainDoctorWindow.add(startDoctorReceptionButton, "wrap 15, al center");  
   startRegistryMainDoctorWindow.add(startDoctorPatientButton, "al center");  
    
    
   receptionDoctorWindow.setBackground(new Color(255,255,255));  
   receptionDoctorWindow.setLayout(new MigLayout());  
   receptionStartDoctorLabelText.setText(doctor.getSecondName() + " " + doctor.getFirstName() + " " + doctor.getThirdName());  
   receptionStartDoctorTableNumberLabelText.setText(String.*valueOf*(doctor.getTableNumber()));  
   receptionStartDoctorSpecializationLabelText.setText(doctor.getSpecialization());  
    
   receptionDoctorWindow.add(receptionStartDoctorLabel, "wrap 5, al left");  
   receptionDoctorWindow.add(receptionStartDoctorLabelText, "wrap 10, al left");  
   receptionDoctorWindow.add(receptionStartDoctorTableNumberLabel, "split 2, al left");  
   receptionDoctorWindow.add(receptionStartDoctorTableNumberLabelText, "wrap 10");  
   receptionDoctorWindow.add(receptionStartDoctorSpecializationLabel, "wrap 5, al left");  
   receptionDoctorWindow.add(receptionStartDoctorSpecializationLabelText, "wrap 10, al left");  
   receptionDoctorWindow.add(receptionBackToReceptionButton, "al left");  
    
    
    
   receptionDoctorWindow2.setBackground(new Color(255,255,255));  
   receptionDoctorWindow2.setLayout(new MigLayout());  
   receptionStartDoctorLabelText2.setText(doctor.getSecondName() + " " + doctor.getFirstName() + " " + doctor.getThirdName());  
   receptionStartDoctorTableNumberLabelText2.setText(String.*valueOf*(doctor.getTableNumber()));  
   receptionStartDoctorSpecializationLabelText2.setText(doctor.getSpecialization());  
    
   receptionDoctorWindow2.add(receptionStartDoctorLabel2, "wrap 5, al left");  
   receptionDoctorWindow2.add(receptionStartDoctorLabelText2, "wrap 10, al left");  
   receptionDoctorWindow2.add(receptionStartDoctorTableNumberLabel2, "split 2, al left");  
   receptionDoctorWindow2.add(receptionStartDoctorTableNumberLabelText2, "wrap 10");  
   receptionDoctorWindow2.add(receptionStartDoctorSpecializationLabel2, "wrap 5, al left");  
   receptionDoctorWindow2.add(receptionStartDoctorSpecializationLabelText2, "wrap 10, al left");  
   receptionDoctorWindow2.add(receptionBackToReceptionButton2, "al left");  
  /////////////////Рабочая панель(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
    
   propertiesAndListOfReceptionPanel.setBackground(new Color(255,255,255));  
   propertiesAndListOfReceptionPanel.setLayout(new MigLayout());  
    
   listOfReceptionPanel.setBackground(new Color(255,255,255));  
   listOfReceptionPanel.setLayout(new MigLayout());  
    
   propertiesListOfReceptionPanel.setBackground(new Color(100,100,100));  
   propertiesListOfReceptionPanel.setLayout(new MigLayout());  
    
   propertiesAndListOfReceptionPanel.add(listOfReceptionPanel, "pushx, growx, h 600!, wrap");  
   propertiesAndListOfReceptionPanel.add(propertiesListOfReceptionPanel, "growx, pushx, h 60!");  
    
   protectListOfReceptionButton.setEnabled(false);  
    
   propertiesListOfReceptionPanel.add(protectListOfReceptionButton, "gaptop 10, al center");  
    
   tableOfReceptionToDoctor.setModel(rtdtm);  
   tableOfReceptionToDoctor.setRowSorter(rowSorter); // Для сортировки по столбцам  
   tableOfReceptionToDoctor.setColumnSelectionAllowed(false);  
   tableOfReceptionToDoctor.getTableHeader().setReorderingAllowed(false);  
   try { rtdtm.addData(jdbcConnection, doctor.getTableNumber()); } // Добавление в модель таблицы данных из БД  
   catch (SQLException e) { e.printStackTrace(); }  
   rtdtm.setColumnStaticWidth(tableOfReceptionToDoctor); // Установка ширины полей  
   tableOfReceptionToDoctor.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*); // Для добавления полосы прокрутки влево-вправо  
    
   listOfReceptionPanel.add(scrollTableOfReceptionToDoctor, "push, grow, al center");  
    
    
   propertiesAndListOfReceptionPanel2.setBackground(new Color(255,255,255));  
   propertiesAndListOfReceptionPanel2.setLayout(new MigLayout());  
    
   listOfReceptionPanel2.setBackground(new Color(255,255,255));  
   listOfReceptionPanel2.setLayout(new MigLayout());  
    
   propertiesListOfReceptionPanel2.setBackground(new Color(100,100,100));  
   propertiesListOfReceptionPanel2.setLayout(new MigLayout());  
    
   propertiesAndListOfReceptionPanel2.add(listOfReceptionPanel2, "pushx, growx, h 600!, wrap");  
   propertiesAndListOfReceptionPanel2.add(propertiesListOfReceptionPanel2, "growx, pushx, h 60!");  
    
   editListOfReceptionButton2.setEnabled(false);  
    
   propertiesListOfReceptionPanel2.add(editListOfReceptionButton2, "gaptop 10, al center");  
    
   tableOfReceptionToDoctor2.setModel(rtdtm2);  
   tableOfReceptionToDoctor2.setRowSorter(rowSorter2); // Для сортировки по столбцам  
   tableOfReceptionToDoctor2.setColumnSelectionAllowed(false);  
   tableOfReceptionToDoctor2.getTableHeader().setReorderingAllowed(false);  
   try { rtdtm2.addData(jdbcConnection, doctor.getTableNumber()); } // Добавление в модель таблицы данных из БД  
   catch (SQLException e) { e.printStackTrace(); }  
   rtdtm2.setColumnStaticWidth(tableOfReceptionToDoctor2); // Установка ширины полей  
   tableOfReceptionToDoctor2.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*); // Для добавления полосы прокрутки влево-вправо  
    
   listOfReceptionPanel2.add(scrollTableOfReceptionToDoctor2, "push, grow, al center");  
    
  /////////////////registryMainWindowPanels(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   doctorTableRightPanel.setBackground(new Color(255,255,0));  
   doctorTableRightPanel.setSize(200, (int) screenSize.getHeight());  
   doctorTableLeftPanel.setBackground(new Color(255,0,0));  
   doctorTableLeftPanel.setSize(1166, (int) screenSize.getHeight());  
    
   mainDoctorWindowPanel.setLayout(new MigLayout());  
   mainDoctorWindowPanel.add(doctorTableLeftPanel, "w 250!, pushy, growy");  
   mainDoctorWindowPanel.add(doctorTableRightPanel, "pushx, pushy, growx, growy");  
  /////////////////registryMainWindowPanels(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
    
   addRegistryMainWindow(scrollMainDoctorWindowPanel, (int)(screenSize.getWidth()), (int)(screenSize.getHeight()));  
    
  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   CardLayout cl = new CardLayout();  
   doctorTableLeftPanel.setLayout(cl);  
   doctorTableLeftPanel.add(startRegistryMainDoctorWindow, "startPage");  
   doctorTableLeftPanel.add(receptionDoctorWindow, "receptionBack");  
   doctorTableLeftPanel.add(receptionDoctorWindow2, "receptionBack2");  
   cl.show(doctorTableLeftPanel, "startPage");  
    
   CardLayout cl2 = new CardLayout();  
   doctorTableRightPanel.setLayout(cl2);  
   doctorTableRightPanel.add(scrollPropertyAndListOfReceptionPanel, "listOfReception");  
   doctorTableRightPanel.add(scrollPropertyAndListOfReceptionPanel2, "listOfReception2");  
   doctorTableRightPanel.add(voidPanel, "voidPanel");  
   cl2.show(doctorTableRightPanel, "voidPanel");  
  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
    
  ////////////////////ActionListeners(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   startDoctorPatientButton.addActionListener(actionEvent -> {  
   cl.show(doctorTableLeftPanel, "receptionBack2");  
   cl2.show(doctorTableRightPanel, "listOfReception2");  
   });  
   startDoctorReceptionButton.addActionListener(actionEvent -> {  
   cl.show(doctorTableLeftPanel, "receptionBack");  
   cl2.show(doctorTableRightPanel, "listOfReception");  
   });  
   receptionBackToReceptionButton.addActionListener(actionEvent -> {  
   cl.show(doctorTableLeftPanel, "startPage");  
   cl2.show(doctorTableRightPanel, "voidPanel");  
   tableOfReceptionToDoctor.clearSelection();  
   protectListOfReceptionButton.setEnabled(false);  
   });  
   receptionBackToReceptionButton2.addActionListener(actionEvent -> {  
   cl.show(doctorTableLeftPanel, "startPage");  
   cl2.show(doctorTableRightPanel, "voidPanel");  
   tableOfReceptionToDoctor2.clearSelection();  
   editListOfReceptionButton2.setEnabled(false);  
   });  
   protectListOfReceptionButton.addActionListener(actionEvent -> {  
   new complaintPatients(this, tableOfReceptionToDoctor.getSelectedRows(),  
   tableOfReceptionToDoctor, tableOfReceptionToDoctor2, protectListOfReceptionButton, rtdtm,  
   rtdtm2, doctor.getSecondName() + " " + doctor.getFirstName() + " " + doctor.getThirdName(),  
   doctor.getTableNumber(), "", "");  
  // try {  
  // rtdtm.addData(jdbcConnection, doctor.getTableNumber());  
  // rtdtm2.addData(jdbcConnection, doctor.getTableNumber());  
  // } catch (SQLException e) {  
  // e.printStackTrace();  
  // }  
  // tableOfReceptionToDoctor.clearSelection();  
  // protectListOfReceptionButton.setEnabled(false);  
  // rtdtm.fireTableDataChanged();  
  // rtdtm2.fireTableDataChanged();  
   });  
   editListOfReceptionButton2.addActionListener(actionEvent -> {  
   String complaint = jdbcConnection.getComplaint(tableOfReceptionToDoctor2.getSelectedRows(), tableOfReceptionToDoctor2);  
   String recomendation = jdbcConnection.getRecomendation(tableOfReceptionToDoctor2.getSelectedRows(), tableOfReceptionToDoctor2);  
   new complaintPatients(this, tableOfReceptionToDoctor2.getSelectedRows(),  
   tableOfReceptionToDoctor2, tableOfReceptionToDoctor, editListOfReceptionButton2, rtdtm,  
   rtdtm2, doctor.getSecondName() + " " + doctor.getFirstName() + " " + doctor.getThirdName(),  
   doctor.getTableNumber(), complaint, recomendation);  
   });  
    
   tableOfReceptionToDoctor.setSelectionModel(new DefaultListSelectionModel() // Модель выбора для удобного интерфейса  
   {  
   @Override  
   public void setSelectionInterval(int startIndex, int endIndex) {  
   if (startIndex == endIndex) // Если выбран один элемент  
   {  
   if (getMinSelectionIndex() != getMaxSelectionIndex())  
   {  
   clearSelection(); // Убрать выделение  
   protectListOfReceptionButton.setEnabled(false);  
   }  
   if (isSelectedIndex(startIndex))  
   {  
   clearSelection(); // Убрать выделение  
   protectListOfReceptionButton.setEnabled(false);  
   }  
   else  
   {  
   super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
   protectListOfReceptionButton.setEnabled(true);  
   }  
   }  
   // Если выбраны многие элементы  
   else {  
   super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
   protectListOfReceptionButton.setEnabled(false);  
   }  
   }  
   });  
   tableOfReceptionToDoctor2.setSelectionModel(new DefaultListSelectionModel() // Модель выбора для удобного интерфейса  
   {  
   @Override  
   public void setSelectionInterval(int startIndex, int endIndex) {  
   if (startIndex == endIndex) // Если выбран один элемент  
   {  
   if (getMinSelectionIndex() != getMaxSelectionIndex())  
   {  
   clearSelection(); // Убрать выделение  
   editListOfReceptionButton2.setEnabled(false);  
   }  
   if (isSelectedIndex(startIndex))  
   {  
   clearSelection(); // Убрать выделение  
   editListOfReceptionButton2.setEnabled(false);  
   }  
   else  
   {  
   super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
   editListOfReceptionButton2.setEnabled(true);  
   }  
   }  
   // Если выбраны многие элементы  
   else {  
   super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
   editListOfReceptionButton2.setEnabled(false);  
   }  
   }  
   });  
  ////////////////////ActionListeners(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
   }  
    
   private void addRegistryMainWindow(JScrollPane \_panel, int width, int height)  
   {  
   this.setBounds(0, 0, width, height); // Устанавливаем оптимальные размеры окна, учитывая размеры экрана  
   this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*); // При нажатии на X произойдёт выход из приложения  
   this.setFocusable(true); // Фокус на окне  
   this.setVisible(true); // Делаем окно видимым  
   this.setResizable(true); // Запрет на изменение размеров окна - нет  
   this.setExtendedState(this.*MAXIMIZED\_BOTH*); // Растяжение на весь экран  
   this.setTitle("Врач"); // Заголовок окна  
    
   this.add(\_panel);  
   }  
  }
* КЛАСС registryMainAdminWindow

package registryApplication;  
  
import Connector.CONST;  
import Connector.JDBCConnection;  
import Users.Admin;  
import Users.Doctor;  
import Users.Patient;  
import com.toedter.calendar.JDateChooser;  
import net.miginfocom.swing.MigLayout;  
import registryApplication.searhModel.searchPatients;  
import registryApplication.searhModel.searchTimetable;  
import registryApplication.tableModels.\*;  
import registryApplication.tableModels.editorDoctorsTable.editorDoctorsTableModel;  
import registryApplication.searhModel.myRowFilter;  
import registryApplication.searhModel.searchDoctors;  
import registryApplication.tableModels.editorPatientModel.editorPatientsTableModel;  
import registryApplication.tableModels.editorTimetableTable.editorTimetableTableModel;  
  
import javax.swing.\*;  
import javax.swing.event.ListSelectionEvent;  
import javax.swing.event.ListSelectionListener;  
import javax.swing.table.AbstractTableModel;  
import javax.swing.table.TableRowSorter;  
import java.awt.\*;  
import java.awt.event.ItemEvent;  
import java.awt.event.ItemListener;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.Scanner;  
import java.util.Vector;  
  
public class registryMainAdminWindow extends JFrame  
{  
 private JDBCConnection jdbcConnection = new JDBCConnection();  
 private Dimension screenSize = java.awt.Toolkit.*getDefaultToolkit*().getScreenSize(); // Создаём переменную, в которой хранится размер экрана  
  
  
 private JPanel mainAdminWindowPanel = new JPanel(); // Основная панель  
 private JScrollPane scrollMainAdminWindowPanel = new JScrollPane(mainAdminWindowPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel adminTablePanel = new JPanel(); // Табличная панель  
  
 private JPanel cardPanel = new JPanel(); // Рабочая панель  
  
///////////////Рабочая панель(begin)///////////////////////////////////////////////////////////////////////////////////////////////////  
 private JPanel voidReferenceAdminTimetablePanel = new JPanel();  
 private JLabel voidReferenceAdminTimetableLabel = new JLabel("Администратор: ");  
 private JLabel voidReferenceTimetableAdminNLPLabel = new JLabel();  
 private JButton backToReferenceFromVoidTimetablePanelButton = new JButton("< Назад");  
  
 private JPanel voidReferenceListOfDoctorsPanel = new JPanel();  
 private JLabel voidReferenceListOfDoctorsAdminLabel = new JLabel("Администратор: ");  
 private JLabel voidReferenceListOfDoctorsAdminNLPLabel = new JLabel();  
 private JButton backToReferenceFromVoidListOfDoctorsPanelButton = new JButton("< Назад");  
  
 private JPanel voidReferenceListOfPatientsPanel = new JPanel();  
 private JLabel voidReferenceListOfPatientsAdminLabel = new JLabel("Администратор: ");  
 private JLabel voidReferenceListOfPatientsAdminNLPLabel = new JLabel();  
 private JButton backToReferenceFromVoidListOfPatientsPanelButton = new JButton("< Назад");  
  
 private JPanel startRegistryMainAdminWindow = new JPanel();  
 private JLabel startAdminLabel = new JLabel("Администратор: ");  
 private JLabel startAdminNLPLabel = new JLabel();  
 private JButton startAdminReferenceButton = new JButton("Справочная");  
 private JButton startAdminServiceButton = new JButton("Служебная");  
 private JButton startAdminNewDataButton = new JButton("Новые данные");  
  
 private JPanel referenceRegistryMainAdminWindow = new JPanel();  
 private JLabel referenceAdminLabel = new JLabel("Администратор: ");  
 private JLabel referenceAdminNLPLabel = new JLabel();  
 private JButton backToStartFromReferenceRegistryMainAdminWindow = new JButton("< Назад");  
 private JButton referenceAdminTimetableButton = new JButton("Расписание");  
 private JButton referenceAdminListDoctorButton = new JButton("Список врачей");  
 private JButton referenceAdminListPatientButton = new JButton("Список пациентов");  
  
 private JPanel serviceRegistryMainAdminWindow = new JPanel();  
 private JLabel serviceAdminLabel = new JLabel("Администратор: ");  
 private JLabel serviceAdminNLPLabel = new JLabel();  
 private JButton backToStartFromServiceRegistryMainAdminWindow = new JButton("< Назад");  
 private JButton serviceAdminDoctorVisitButton = new JButton("Назначить приём");  
  
 private JPanel voidNewDataRegistryMainAdminWindow = new JPanel();  
 private JLabel voidNewDataAdminLabel = new JLabel("Администратор: ");  
 private JLabel voidNewDataAdminNLPLabel = new JLabel();  
 private JButton voidBackToStartFromNewDataRegistryMainAdminWindow = new JButton("< Назад");  
  
 private JPanel newDataRegistryMainAdminWindow = new JPanel();  
 private JLabel newDataAdminLabel = new JLabel("Администратор: ");  
 private JLabel newDataAdminNLPLabel = new JLabel();  
 private JButton backToStartFromNewDataRegistryMainAdminWindow = new JButton("< Назад");  
 private JButton newDataAdminAddDoctorButton = new JButton("Добавить врача");  
 private JButton newDataAdminAddPatientButton = new JButton("Добавить пациента");  
///////////////Рабочая панель(end)///////////////////////////////////////////////////////////////////////////////////////////////////  
  
///////////////Табличная панель(begin)///////////////////////////////////////////////////////////////////////////////////////////////////  
 private JPanel voidPanel = new JPanel();  
 /////////////////////registerDoctorPanel(begin)//////////////////////////////////////////////////////////////////////////////////////  
 private JPanel registerDoctorPanel = new JPanel();  
 private JLabel secondNameDoctorLabel = new JLabel("Фамилия: ");  
 private JLabel firstNameDoctorLabel = new JLabel("Имя: ");  
 private JLabel thirdNameDoctorLabel = new JLabel("Отчество: ");  
 private JLabel sexOfDoctorLabel = new JLabel("Пол: ");  
 private JLabel maleSexOfDoctorLabel = new JLabel("Муж");  
 private JLabel femaleSexOfDoctorLabel = new JLabel("Жен");  
 private JLabel birthdayDayDoctorLabel = new JLabel("Дата рождения: ");  
 private JLabel numberOfPlotDoctorLabel = new JLabel("Номер участка: ");  
 private JLabel specializationOfDoctorLabel = new JLabel("Специальность: ");  
 private JLabel dateOfStartJobDoctorLabel = new JLabel("Дата приёма на работу: ");  
 private JLabel newPasswordDoctorLabel = new JLabel("Придумайте пароль: ");  
 private JLabel mailAddressDoctorLabel = new JLabel("Введите почту врача: ");  
 private JLabel addressOfLiveDoctorLabel = new JLabel("Адрес проживания: ");  
 private JLabel cityOfDoctorLabel = new JLabel("Город: ");  
 private JLabel streetOfDoctorLabel = new JLabel("Улица: ");  
 private JLabel buildingOfDoctorLabel = new JLabel("Дом: ");  
 private JLabel apartmentOfDoctorLabel = new JLabel("Квартира: ");  
 private JTextField secondNameDoctorText = new JTextField();  
 private JTextField firstNameDoctorText = new JTextField();  
 private JTextField thirdNameDoctorText = new JTextField();  
 private JCheckBox maleDoctorCheckBox = new JCheckBox();  
 private JCheckBox femaleDoctorCheckBox = new JCheckBox();  
 private JDateChooser birthdayDayDoctorTable = new JDateChooser();  
 private JComboBox<Integer> numberOfPlotDoctorComboBox;  
 private JComboBox<String> specializationOfDoctorComboBox;  
 private JDateChooser dateOfStartJobDoctorDate = new JDateChooser();  
 private JTextField newPasswordDoctorText = new JTextField();  
 private JTextField mailAddressDoctorText = new JTextField();  
 private JTextField cityOfDoctorText = new JTextField();  
 private JTextField streetOfDoctorText = new JTextField();  
 private JTextField buildingOfDoctorText = new JTextField();  
 private JTextField apartmentOfDoctorText = new JTextField();  
 private JLabel notDoctorData = new JLabel("Укажите все данные!!!");  
 private JButton registerDoctorButton = new JButton("Зарегистрировать");  
 private JScrollPane scrollRegisterDoctor = new JScrollPane(registerDoctorPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
 /////////////////////registerDoctorPanel(end)////////////////////////////////////////////////////////////////////////////////////////  
  
 /////////////////////registerPatientPanel(begin)//////////////////////////////////////////////////////////////////////////////////////  
 private JPanel registerPatientPanel = new JPanel();  
 private JLabel secondNamePatientLabel = new JLabel("Фамилия: ");  
 private JLabel firstNamePatientLabel = new JLabel("Имя: ");  
 private JLabel thirdNamePatientLabel = new JLabel("Отчество: ");  
 private JLabel sexOfPatientLabel = new JLabel("Пол: ");  
 private JLabel maleSexOfPatientLabel = new JLabel("Муж");  
 private JLabel femaleSexOfPatientLabel = new JLabel("Жен");  
 private JLabel birthdayDayPatientLabel = new JLabel("Дата рождения: ");  
 private JLabel dateOfCreatePatientCardLabel = new JLabel("Дата создания карточки: ");  
 private JLabel numberOfPassportPatientLabel = new JLabel("Номер паспорта: ");  
 private JLabel numberOfPhonePatientLabel = new JLabel("Номер телефона: ");  
 private JLabel numberOfInsurancePolicyLabel = new JLabel("Номер страхового полиса: ");  
 private JLabel addressOfLivePatientLabel = new JLabel("Адрес проживания: ");  
 private JLabel cityOfPatientLabel = new JLabel("Город: ");  
 private JLabel streetOfPatientLabel = new JLabel("Улица: ");  
 private JLabel buildingOfPatientLabel = new JLabel("Дом: ");  
 private JLabel apartmentOfPatientLabel = new JLabel("Квартира: ");  
 private JTextField secondNamePatientText = new JTextField();  
 private JTextField firstNamePatientText = new JTextField();  
 private JTextField thirdNamePatientText = new JTextField();  
 private JCheckBox malePatientCheckBox = new JCheckBox();  
 private JCheckBox femalePatientCheckBox = new JCheckBox();  
 private JDateChooser birthdayDayPatientTable = new JDateChooser();  
 private JDateChooser dateOfCreatePatientCard = new JDateChooser();  
 private JTextField numberOfPassportPatientText = new JTextField();  
 private JTextField numberOfPhonePatientText = new JTextField();  
 private JTextField numberOfInsurancePolicyText = new JTextField();  
 private JTextField cityOfPatientText = new JTextField();  
 private JList<String> streetOfPatientText;  
 private JTextField buildingOfPatientText = new JTextField();  
 private JTextField apartmentOfPatientText = new JTextField();  
 private JLabel notPatientData = new JLabel("Укажите все данные!!!");  
 private JButton registerPatientButton = new JButton("Зарегистрировать");  
 private JScrollPane scrollRegisterPatient = new JScrollPane(registerPatientPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
 /////////////////////registerPatientPanel(end)////////////////////////////////////////////////////////////////////////////////////////  
  
 /////////////////////listOfDoctorsPanel(begin)//////////////////////////////////////////////////////////////////////////////////////  
 private JPanel propertiesAndListOfDoctorsPanel = new JPanel();  
 private JScrollPane scrollPropertyAndListOfDoctorsPanel = new JScrollPane(propertiesAndListOfDoctorsPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel listOfDoctorsPanel = new JPanel();  
 private JTable tableOfDoctors = new JTable();  
 private doctorsTableModel dtm = new doctorsTableModel(tableOfDoctors);  
 private TableRowSorter<AbstractTableModel> rowSorter = new TableRowSorter(dtm);  
 private JScrollPane scrollTableOfDoctors = new JScrollPane(tableOfDoctors, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel propertiesListOfDoctorPanel = new JPanel();  
 private JButton editListOfDoctorsButton = new JButton("Редактировать");  
 private JButton deleteRowsOfListOfDoctorsButton = new JButton("Удалить");  
 private JButton searchListOfDoctorsButton = new JButton("Поиск");  
 private JButton throwOffSearchListOfDoctorButton = new JButton("Сбросить поиск");  
 private JLabel sortInfo = new JLabel("Для сортировки дважды щёлкните по заголовку необходимой колонки");  
  
 private JPanel propertiesAndEditListOfDoctorsPanel = new JPanel();  
 private JScrollPane scrollPropertyAndEditListOfDoctorsPanel = new JScrollPane(propertiesAndEditListOfDoctorsPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel editPropertiesListOfDoctorPanel = new JPanel();  
 private JButton applyEditListOfDoctorsButton = new JButton("Применить");  
 private JButton cancelEditListOfDoctorsButton = new JButton("Отмена");  
  
 private JPanel editorListOfDoctorsPanel = new JPanel();  
 private JTable editorTableOfDoctors = new JTable();  
 private editorDoctorsTableModel edtm = new editorDoctorsTableModel(this, editorTableOfDoctors, applyEditListOfDoctorsButton, dtm.getColumnTypes(), dtm.getColumnNames());  
 private JScrollPane scrollEditorTableOfDoctors = new JScrollPane(editorTableOfDoctors, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
/////////////////////listOfDoctorsPanel(end)////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////////listOfPatientPanel(begin)//////////////////////////////////////////////////////////////////////////////////////  
 private JPanel propertiesAndListOfPatientPanel = new JPanel();  
 private JScrollPane scrollPropertyAndListOfPatientsPanel = new JScrollPane(propertiesAndListOfPatientPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel listOfPatientsPanel = new JPanel();  
 private JTable tableOfPatients = new JTable();  
 private patientsTableModel ptm = new patientsTableModel(tableOfPatients);  
 private TableRowSorter<AbstractTableModel> rowSorterPatient = new TableRowSorter(ptm);  
 private JScrollPane scrollTableOfPatients = new JScrollPane(tableOfPatients, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel propertiesListOfPatientPanel = new JPanel();  
 private JButton editListOfPatientsButton = new JButton("Редактировать");  
 private JButton deleteRowsOfListOfPatientsButton = new JButton("Удалить");  
 private JButton searchListOfPatientsButton = new JButton("Поиск");  
 private JButton throwOffSearchListOfPatientButton = new JButton("Сбросить поиск");  
 private JLabel sortInfoPatient = new JLabel("Для сортировки дважды щёлкните по заголовку необходимой колонки");  
  
 private JPanel propertiesAndEditListOfPatientsPanel = new JPanel();  
 private JScrollPane scrollPropertyAndEditListOfPatientsPanel = new JScrollPane(propertiesAndEditListOfPatientsPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel editPropertiesListOfPatientPanel = new JPanel();  
 private JButton applyEditListOfPatientsButton = new JButton("Применить");  
 private JButton cancelEditListOfPatientsButton = new JButton("Отмена");  
  
 private JPanel editorListOfPatientsPanel = new JPanel();  
 private JTable editorTableOfPatients = new JTable();  
 private editorPatientsTableModel eptm = new editorPatientsTableModel(this, editorTableOfPatients, applyEditListOfPatientsButton, ptm.getColumnTypes(), ptm.getColumnNames(), ptm.getColumnRUSNames());  
 private JScrollPane scrollEditorTableOfPatients = new JScrollPane(editorTableOfPatients, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
/////////////////////listOfPatientPanel(end)////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////////timetablePanel(begin)//////////////////////////////////////////////////////////////////////////////////////  
 private JPanel propertiesAndTimetablePanel = new JPanel();  
 private JScrollPane scrollPropertiesAndTimetablePanel = new JScrollPane(propertiesAndTimetablePanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel timetablePanel = new JPanel();  
 private JTable tableOfTimetable = new JTable();  
 private timetableTableModel ttm;  
 private TableRowSorter<AbstractTableModel> rowSorterTimetable;  
 private JScrollPane scrollTableOfTimetable = new JScrollPane(tableOfTimetable, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel propertiesTimetablePanel = new JPanel();  
 private JButton editTimetableButton = new JButton("Редактировать");  
 private JButton searchTimetableButton = new JButton("Поиск");  
 private JButton throwOffSearchTimetableButton = new JButton("Сбросить поиск");  
 private JButton previousWeekButton = new JButton("< Предыдущая неделя");  
 private JButton nextWeekButton = new JButton("Следующая неделя >");  
  
  
 private JPanel propertiesAndEditTimetablePanel = new JPanel();  
 private JScrollPane scrollPropertyAndEditTimetablePanel = new JScrollPane(propertiesAndEditTimetablePanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel editPropertiesTimetablePanel = new JPanel();  
 private JButton applyEditTimetableButton = new JButton("Применить");  
 private JButton cancelEditTimetableButton = new JButton("Отмена");  
  
 private JPanel editorTimetablePanel = new JPanel();  
 private JTable editorTableOfTimetable = new JTable();  
 private editorTimetableTableModel ettm;  
 private JScrollPane scrollEditorTableOfTimetable = new JScrollPane(editorTableOfTimetable, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
/////////////////////timetablePanel(end)////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////////receptionPanel(begin)////////////////////////////////////////////////////////////////////////////////////////  
 //////////////////////receptionVoidPanel(begin)///////////////////////////////////////////////////////////////////////////////  
 private JPanel voidServiceAdminReceptionPanel = new JPanel();  
 private JLabel voidServiceAdminReceptionLabel = new JLabel("Администратор: ");  
 private JLabel voidServiceReceptionAdminNLPLabel = new JLabel();  
 private JButton backToServiceFromVoidReceptionPanelButton = new JButton("< Назад");  
 //////////////////////receptionVoidPanel(end)/////////////////////////////////////////////////////////////////////////////////  
  
 //////////////////////receptionPanelMain(begin)///////////////////////////////////////////////////////////////////////////////  
 private JPanel receptionPanel = new JPanel();  
 private JScrollPane scrollReceptionPanel= new JScrollPane(receptionPanel, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel receptionTablePanel = new JPanel();  
 private JTable tableOfReception = new JTable();  
 private receptionTableModel rtm;  
 private TableRowSorter<AbstractTableModel> rowSorterReception;  
 private JScrollPane scrollTableOfReception = new JScrollPane(tableOfReception, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JPanel propertiesReceptionPanel = new JPanel();  
 private JLabel chooseSpecializationLabel = new JLabel("Специальность: ");  
 private JComboBox chooseSpecializationJComboBox;  
  
 private JLabel chooseDoctorLabel = new JLabel("Доктор: ");  
 private DefaultListModel<String> chooseDoctorJListModel = new DefaultListModel<>();  
 private JList<String> chooseDoctorJList = new JList(chooseDoctorJListModel);  
 private JScrollPane scrollChooseDoctorJList = new JScrollPane(chooseDoctorJList, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private DefaultComboBoxModel chooseDateJComboBoxModel = new DefaultComboBoxModel();  
 private JLabel chooseDateLabel = new JLabel("Дата: ");  
 private JComboBox chooseDateJComboBox = new JComboBox(chooseDateJComboBoxModel);  
  
 private JLabel showHoursLabel = new JLabel("Часы приёма: ");  
 private JTextArea showHoursText = new JTextArea();  
  
 private DefaultComboBoxModel chooseTimeJComboBoxModel = new DefaultComboBoxModel();  
 private JLabel chooseTimeLabel = new JLabel("Свободное время: ");  
 private JComboBox chooseTimeJComboBox = new JComboBox(chooseTimeJComboBoxModel);  
  
 private JLabel choosePatientLabel = new JLabel("Пациент: ");  
 private DefaultListModel<String> choosePatientJListModel = new DefaultListModel<>();  
 private JList<String> choosePatientJList = new JList(choosePatientJListModel);  
 private JScrollPane scrollChoosePatientJList = new JScrollPane(choosePatientJList, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 private JButton addToReceptionButton = new JButton("Записать на приём");  
 private JButton cancelFromReceptionButton = new JButton("Отменить запись(-и)");  
 //////////////////////receptionPanelMain(edn)/////////////////////////////////////////////////////////////////////////////////  
/////////////////////receptionPanel(end)//////////////////////////////////////////////////////////////////////////////////////////  
  
///////////////Табличная панель(end)///////////////////////////////////////////////////////////////////////////////////////////////////  
  
 public registryMainAdminWindow(Admin admin) // Конструктор класса без параметров  
 {  
/////////////////newDataRegistryMainAdminWindow(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 ////////////////////////newDataRegistryMainAdminWindowSource(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
 newDataRegistryMainAdminWindow.setBackground(new Color(255,255,255));  
 newDataRegistryMainAdminWindow.setLayout(new MigLayout());  
 newDataAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 newDataRegistryMainAdminWindow.add(newDataAdminLabel, "al left, wrap 5");  
 newDataRegistryMainAdminWindow.add(newDataAdminNLPLabel, "al left, wrap 5");  
 newDataRegistryMainAdminWindow.add(backToStartFromNewDataRegistryMainAdminWindow, "al left, wrap 35");  
 newDataRegistryMainAdminWindow.add(newDataAdminAddDoctorButton, "al center, gapleft 50, wrap 15");  
 newDataRegistryMainAdminWindow.add(newDataAdminAddPatientButton, "al center, gapleft 50");  
 ////////////////////////newDataRegistryMainAdminWindowSource(end)////////////////////////////////////////////////////////////////////////////////////////////////  
  
 ////////////////////////voidNewDataRegistryMainAdminWindow(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
 voidNewDataRegistryMainAdminWindow.setBackground(new Color(255,255,255));  
 voidNewDataRegistryMainAdminWindow.setLayout(new MigLayout());  
 voidNewDataAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 voidNewDataRegistryMainAdminWindow.add(voidNewDataAdminLabel, "al left, wrap 5");  
 voidNewDataRegistryMainAdminWindow.add(voidNewDataAdminNLPLabel, "al left, wrap 5");  
 voidNewDataRegistryMainAdminWindow.add(voidBackToStartFromNewDataRegistryMainAdminWindow, "al left");  
 ////////////////////////voidNewDataRegistryMainAdminWindow(end)////////////////////////////////////////////////////////////////////////////////////////////////  
/////////////////newDataRegistryMainAdminWindow(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////serviceRegistryMainAdminWindow(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 serviceRegistryMainAdminWindow.setBackground(new Color(255,255,255));  
 serviceRegistryMainAdminWindow.setLayout(new MigLayout());  
 serviceAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 serviceRegistryMainAdminWindow.add(serviceAdminLabel, "al left, wrap 5");  
 serviceRegistryMainAdminWindow.add(serviceAdminNLPLabel, "al left, wrap 5");  
 serviceRegistryMainAdminWindow.add(backToStartFromServiceRegistryMainAdminWindow, "al left, wrap 35");  
 serviceRegistryMainAdminWindow.add(serviceAdminDoctorVisitButton, "al center, gapleft 50");  
/////////////////serviceRegistryMainAdminWindow(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////referenceRegistryMainAdminWindow(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 ////////////////////////referenceSource(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
 referenceRegistryMainAdminWindow.setBackground(new Color(255,255,255));  
 referenceRegistryMainAdminWindow.setLayout(new MigLayout());  
 referenceAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 referenceRegistryMainAdminWindow.add(referenceAdminLabel, "al left, wrap 5");  
 referenceRegistryMainAdminWindow.add(referenceAdminNLPLabel, "al left, wrap 5");  
 referenceRegistryMainAdminWindow.add(backToStartFromReferenceRegistryMainAdminWindow, "al left, wrap 35");  
 referenceRegistryMainAdminWindow.add(referenceAdminTimetableButton, "al center, gapleft 50, wrap 15");  
 referenceRegistryMainAdminWindow.add(referenceAdminListDoctorButton, "al center, gapleft 50, wrap 15");  
 referenceRegistryMainAdminWindow.add(referenceAdminListPatientButton, "al center, gapleft 50");  
 ////////////////////////referenceSource(end)////////////////////////////////////////////////////////////////////////////////////////////////  
  
 ////////////////////////voidReferenceListOfDoctorsPanel(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
 voidReferenceListOfDoctorsPanel.setBackground(new Color(255,255,255));  
 voidReferenceListOfDoctorsPanel.setLayout(new MigLayout());  
 voidReferenceListOfDoctorsAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 voidReferenceListOfDoctorsPanel.add(voidReferenceListOfDoctorsAdminLabel, "al left, wrap 5");  
 voidReferenceListOfDoctorsPanel.add(voidReferenceListOfDoctorsAdminNLPLabel, "al left, wrap 5");  
 voidReferenceListOfDoctorsPanel.add(backToReferenceFromVoidListOfDoctorsPanelButton, "al left");  
 ////////////////////////voidReferenceListOfDoctorsPanel(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
  
 ////////////////////////voidReferenceListOfDoctorsPanel(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
 voidReferenceListOfPatientsPanel.setBackground(new Color(255,255,255));  
 voidReferenceListOfPatientsPanel.setLayout(new MigLayout());  
 voidReferenceListOfPatientsAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 voidReferenceListOfPatientsPanel.add(voidReferenceListOfPatientsAdminLabel, "al left, wrap 5");  
 voidReferenceListOfPatientsPanel.add(voidReferenceListOfPatientsAdminNLPLabel, "al left, wrap 5");  
 voidReferenceListOfPatientsPanel.add(backToReferenceFromVoidListOfPatientsPanelButton, "al left");  
 ////////////////////////voidReferenceListOfDoctorsPanel(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
  
  
 ////////////////////////voidReferenceTimetablePanel(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
 voidReferenceAdminTimetablePanel.setBackground(new Color(255,255,255));  
 voidReferenceAdminTimetablePanel.setLayout(new MigLayout());  
 voidReferenceTimetableAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 voidReferenceAdminTimetablePanel.add(voidReferenceAdminTimetableLabel, "al left, wrap 5");  
 voidReferenceAdminTimetablePanel.add(voidReferenceTimetableAdminNLPLabel, "al left, wrap 5");  
 voidReferenceAdminTimetablePanel.add(backToReferenceFromVoidTimetablePanelButton, "al left");  
 ////////////////////////voidReferenceTimetablePanel(end)////////////////////////////////////////////////////////////////////////////////////////////////  
  
 ////////////////////////voidReferenceReceptionPanel(begin)//////////////////////////////////////////////////////////////////////////////////////////////  
 voidServiceAdminReceptionPanel.setBackground(new Color(255,255,255));  
 voidServiceAdminReceptionPanel.setLayout(new MigLayout());  
 voidServiceReceptionAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 voidServiceAdminReceptionPanel.add(voidServiceAdminReceptionLabel, "al left, wrap 5");  
 voidServiceAdminReceptionPanel.add(voidServiceReceptionAdminNLPLabel, "al left, wrap 5");  
 voidServiceAdminReceptionPanel.add(backToServiceFromVoidReceptionPanelButton, "al left");  
 ////////////////////////voidReferenceReceptionPanel(end)////////////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////referenceRegistryMainAdminWindow(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////startRegistryMainAdminWindow(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 startRegistryMainAdminWindow.setBackground(new Color(255,255,255));  
 startRegistryMainAdminWindow.setLayout(new MigLayout());  
 startAdminNLPLabel.setText(admin.getSecondName() + " " + admin.getFirstName() + " " + admin.getThirdName());  
  
 startRegistryMainAdminWindow.add(startAdminLabel, "al left, wrap 5");  
 startRegistryMainAdminWindow.add(startAdminNLPLabel, "al left, wrap 66");  
 startRegistryMainAdminWindow.add(startAdminReferenceButton, "al center, gapleft 50, wrap 15");  
 startRegistryMainAdminWindow.add(startAdminServiceButton, "al center, gapleft 50, wrap 15");  
 startRegistryMainAdminWindow.add(startAdminNewDataButton, "al center, gapleft 50");  
/////////////////startRegistryMainAdminWindow(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
  
////////////////voidPanel(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////  
 registerDoctorPanel.setBackground(new Color(122,122,122));  
////////////////voidDoctorPanel(end)///////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
////////////////registerDoctorPanel(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////  
 registerDoctorPanel.setBackground(new Color(255,255,255));  
 registerDoctorPanel.setLayout(new MigLayout());  
 notDoctorData.setForeground(new Color(230,50,50));  
 notDoctorData.setVisible(false);  
  
 Vector<Integer> data = new Vector();  
 for(int i = 1; i <= 37; i++)  
 {  
 data.add(i);  
 }  
 numberOfPlotDoctorComboBox = new JComboBox(data);  
 numberOfPlotDoctorComboBox.setSelectedIndex(0);  
 numberOfPlotDoctorComboBox.setFocusable(false);  
  
 Vector vector = new Vector();  
 vector.addAll(jdbcConnection.getAllSpecialization());  
 specializationOfDoctorComboBox = new JComboBox(vector);  
 specializationOfDoctorComboBox.setSelectedIndex(0);  
 specializationOfDoctorComboBox.setFocusable(false);  
  
 birthdayDayDoctorTable.setDateFormatString("yyyy-MM-dd");  
 birthdayDayDoctorTable.setDate(new Date());  
 dateOfStartJobDoctorDate.setDateFormatString("yyyy-MM-dd");  
 dateOfStartJobDoctorDate.setDate(new Date());  
  
 registerDoctorPanel.add(secondNameDoctorLabel, "gapleft 250, gaptop 50, al right");  
 registerDoctorPanel.add(secondNameDoctorText, "w 250!, wrap");  
 registerDoctorPanel.add(firstNameDoctorLabel, "gapleft 250, al right");  
 registerDoctorPanel.add(firstNameDoctorText, "w 250!, wrap");  
 registerDoctorPanel.add(thirdNameDoctorLabel, "gapleft 250, al right");  
 registerDoctorPanel.add(thirdNameDoctorText, "w 250!, wrap");  
 registerDoctorPanel.add(sexOfDoctorLabel, "gapleft 250, al right");  
 registerDoctorPanel.add(maleSexOfDoctorLabel, "split 4");  
 registerDoctorPanel.add(maleDoctorCheckBox);  
 registerDoctorPanel.add(femaleSexOfDoctorLabel, "split 2");  
 registerDoctorPanel.add(femaleDoctorCheckBox, "wrap");  
 registerDoctorPanel.add(birthdayDayDoctorLabel, "gapleft 250, al right");  
 registerDoctorPanel.add(birthdayDayDoctorTable, "w 150!, wrap");  
 registerDoctorPanel.add(numberOfPlotDoctorLabel, "gapleft 298, al right");  
 registerDoctorPanel.add(numberOfPlotDoctorComboBox, "wrap");  
 registerDoctorPanel.add(specializationOfDoctorLabel, "gapleft 250, al right");  
 registerDoctorPanel.add(specializationOfDoctorComboBox, "w 300!, wrap");  
 registerDoctorPanel.add(dateOfStartJobDoctorLabel, "gapleft 250, al right");  
 registerDoctorPanel.add(dateOfStartJobDoctorDate, "w 150!, wrap");  
 registerDoctorPanel.add(newPasswordDoctorLabel, "gapleft 250, al right");  
 registerDoctorPanel.add(newPasswordDoctorText, "w 250!, wrap");  
 registerDoctorPanel.add(mailAddressDoctorLabel, "gapleft 250, al right");  
 registerDoctorPanel.add(mailAddressDoctorText, "w 250!, wrap");  
 registerDoctorPanel.add(addressOfLiveDoctorLabel, "gapleft 250, span 1 4, top, al right"); // Растянуть компонент 1х4  
 registerDoctorPanel.add(cityOfDoctorLabel, "gapleft 18, split 2");  
 registerDoctorPanel.add(cityOfDoctorText, "w 250!, wrap");  
 registerDoctorPanel.add(streetOfDoctorLabel, "gapleft 18, split 2");  
 registerDoctorPanel.add(streetOfDoctorText, "w 250!, wrap");  
 registerDoctorPanel.add(buildingOfDoctorLabel, "gapleft 30, split 2");  
 registerDoctorPanel.add(buildingOfDoctorText, "w 250!, wrap");  
 registerDoctorPanel.add(apartmentOfDoctorLabel, "split 2");  
 registerDoctorPanel.add(apartmentOfDoctorText, "w 250!, wrap 20");  
 registerDoctorPanel.add(notDoctorData, "gapleft 250, span 2 1, al center, wrap 25");  
 registerDoctorPanel.add(registerDoctorButton, "gapleft 250, span 2 1, al center");  
////////////////registerDoctorPanel(end)///////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
////////////////registerPatientPanel(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////  
 registerPatientPanel.setBackground(new Color(255,255,255));  
 registerPatientPanel.setLayout(new MigLayout());  
 notPatientData.setForeground(new Color(230,50,50));  
 notPatientData.setVisible(false);  
  
 birthdayDayPatientTable.setDateFormatString("yyyy-MM-dd");  
 birthdayDayPatientTable.setDate(new Date());  
 dateOfCreatePatientCard.setDateFormatString("yyyy-MM-dd");  
 dateOfCreatePatientCard.setDate(new Date());  
  
 streetOfPatientText = new JList(jdbcConnection.getAllStrees());  
 JScrollPane scrollStreetOfPatientText = new JScrollPane(streetOfPatientText, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
  
 registerPatientPanel.add(secondNamePatientLabel, "gapleft 250, gaptop 50, al right");  
 registerPatientPanel.add(secondNamePatientText, "w 250!, wrap");  
 registerPatientPanel.add(firstNamePatientLabel, "gapleft 250, al right");  
 registerPatientPanel.add(firstNamePatientText, "w 250!, wrap");  
 registerPatientPanel.add(thirdNamePatientLabel, "gapleft 250, al right");  
 registerPatientPanel.add(thirdNamePatientText, "w 250!, wrap");  
 registerPatientPanel.add(sexOfPatientLabel, "gapleft 250, al right");  
 registerPatientPanel.add(maleSexOfPatientLabel, "split 4");  
 registerPatientPanel.add(malePatientCheckBox);  
 registerPatientPanel.add(femaleSexOfPatientLabel, "split 2");  
 registerPatientPanel.add(femalePatientCheckBox, "wrap");  
 registerPatientPanel.add(birthdayDayPatientLabel, "gapleft 250, al right");  
 registerPatientPanel.add(birthdayDayPatientTable, "w 150!, wrap");  
 registerPatientPanel.add(dateOfCreatePatientCardLabel, "gapleft 250, al right");  
 registerPatientPanel.add(dateOfCreatePatientCard, "w 150!, wrap");  
 registerPatientPanel.add(numberOfPassportPatientLabel, "gapleft 250, al right");  
 registerPatientPanel.add(numberOfPassportPatientText, "w 150!, wrap");  
 registerPatientPanel.add(numberOfPhonePatientLabel, "gapleft 250, al right");  
 registerPatientPanel.add(numberOfPhonePatientText, "w 200!, wrap");  
 registerPatientPanel.add(numberOfInsurancePolicyLabel, "gapleft 250, al right");  
 registerPatientPanel.add(numberOfInsurancePolicyText, "w 150!, wrap");  
 registerPatientPanel.add(addressOfLivePatientLabel, "gapleft 250, span 1 4, top, al right"); // Растянуть компонент 1х4  
 registerPatientPanel.add(cityOfPatientLabel, "gapleft 18, split 2");  
 registerPatientPanel.add(cityOfPatientText, "w 250!, wrap");  
 registerPatientPanel.add(streetOfPatientLabel, "gapleft 18, split 2");  
 registerPatientPanel.add(scrollStreetOfPatientText, "w 250!, h 100!, wrap");  
 registerPatientPanel.add(buildingOfPatientLabel, "gapleft 30, split 2");  
 registerPatientPanel.add(buildingOfPatientText, "w 250!, wrap");  
 registerPatientPanel.add(apartmentOfPatientLabel, "split 2");  
 registerPatientPanel.add(apartmentOfPatientText, "w 250!, wrap 20");  
 registerPatientPanel.add(notPatientData, "gapleft 250, span 2 1, al center, wrap 25");  
 registerPatientPanel.add(registerPatientButton, "gapleft 250, span 2 1, al center");  
////////////////registerPatientPanel(end)///////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
////////////////listOfPatientPanel(begin)///////////////////////////////////////////////////////////////////////////////////////////////////////////  
 ///////////////////propertiesAndListOfDoctorsPanel(begin)///////////////////////////////////////////////////////////////////////////////////  
 propertiesAndListOfPatientPanel.setBackground(new Color(255,255,255));  
 propertiesAndListOfPatientPanel.setLayout(new MigLayout());  
  
 listOfPatientsPanel.setBackground(new Color(255,255,255));  
 listOfPatientsPanel.setLayout(new MigLayout());  
  
 propertiesListOfPatientPanel.setBackground(new Color(100,100,100));  
 propertiesListOfPatientPanel.setLayout(new MigLayout());  
  
 propertiesAndListOfPatientPanel.add(listOfPatientsPanel, "pushx, growx, h 600!, wrap");  
 propertiesAndListOfPatientPanel.add(propertiesListOfPatientPanel, "growx, pushx, h 60!");  
  
 editListOfPatientsButton.setEnabled(false);  
 deleteRowsOfListOfPatientsButton.setEnabled(false);  
 throwOffSearchListOfPatientButton.setEnabled(false);  
 sortInfoPatient.setForeground(new Color(255,255,255));  
  
 propertiesListOfPatientPanel.add(editListOfPatientsButton, "split 6, gaptop 10, gapleft 45");  
 propertiesListOfPatientPanel.add(deleteRowsOfListOfPatientsButton);  
 propertiesListOfPatientPanel.add(searchListOfPatientsButton, "split 2, span 1 2, gapleft 120");  
 propertiesListOfPatientPanel.add(throwOffSearchListOfPatientButton);  
 propertiesListOfPatientPanel.add(sortInfoPatient, "gapleft 60");  
  
 tableOfPatients.setModel(ptm);  
 tableOfPatients.setRowSorter(rowSorterPatient); // Для сортировки по столбцам  
 tableOfPatients.setColumnSelectionAllowed(false);  
 tableOfPatients.getTableHeader().setReorderingAllowed(false);  
 try { ptm.addData(jdbcConnection); } // Добавление в модель таблицы данных из БД  
 catch (SQLException e) { e.printStackTrace(); }  
 ptm.setColumnStaticWidth(tableOfPatients); // Установка ширины полей  
 tableOfPatients.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*); // Для добавления полосы прокрутки влево-вправо  
  
 listOfPatientsPanel.add(scrollTableOfPatients, "push, grow, al center");  
 ///////////////////propertiesAndListOfDoctorsPanel(end)/////////////////////////////////////////////////////////////////////////////////////  
  
 ///////////////////propertiesAndEditListOfDoctorsPanel(begin)///////////////////////////////////////////////////////////////////////////////////  
 propertiesAndEditListOfPatientsPanel.setBackground(new Color(255,255,255));  
 propertiesAndEditListOfPatientsPanel.setLayout(new MigLayout());  
  
 editorListOfPatientsPanel.setBackground(new Color(255,255,255));  
 editorListOfPatientsPanel.setLayout(new MigLayout());  
  
 editPropertiesListOfPatientPanel.setBackground(new Color(100,100,100));  
 editPropertiesListOfPatientPanel.setLayout(new MigLayout());  
  
 applyEditListOfPatientsButton.setEnabled(false);  
  
 propertiesAndEditListOfPatientsPanel.add(editorListOfPatientsPanel, "pushx, growx, h 600!, wrap");  
 propertiesAndEditListOfPatientsPanel.add(editPropertiesListOfPatientPanel, "growx, pushx, h 60!");  
  
 editPropertiesListOfPatientPanel.add(applyEditListOfPatientsButton, "gaptop 10, split 2, al center");  
 editPropertiesListOfPatientPanel.add(cancelEditListOfPatientsButton, "gaptop 10");  
  
 editorTableOfPatients.setModel(eptm);  
 editorTableOfPatients.setColumnSelectionAllowed(false);  
 editorTableOfPatients.getTableHeader().setReorderingAllowed(false);  
 eptm.setColumnStaticWidth(editorTableOfPatients);  
 editorTableOfPatients.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 editorListOfPatientsPanel.add(scrollEditorTableOfPatients, "push, grow, al center");  
 ///////////////////propertiesAndEditListOfDoctorsPanel(end)/////////////////////////////////////////////////////////////////////////////////////  
////////////////listOfPatientPanel(end)/////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
////////////////listOfDoctorPanel(begin)///////////////////////////////////////////////////////////////////////////////////////////////////////////  
 ///////////////////propertiesAndListOfDoctorsPanel(begin)///////////////////////////////////////////////////////////////////////////////////  
 propertiesAndListOfDoctorsPanel.setBackground(new Color(255,255,255));  
 propertiesAndListOfDoctorsPanel.setLayout(new MigLayout());  
  
 listOfDoctorsPanel.setBackground(new Color(255,255,255));  
 listOfDoctorsPanel.setLayout(new MigLayout());  
  
 propertiesListOfDoctorPanel.setBackground(new Color(100,100,100));  
 propertiesListOfDoctorPanel.setLayout(new MigLayout());  
  
 propertiesAndListOfDoctorsPanel.add(listOfDoctorsPanel, "pushx, growx, h 600!, wrap");  
 propertiesAndListOfDoctorsPanel.add(propertiesListOfDoctorPanel, "growx, pushx, h 60!");  
  
 editListOfDoctorsButton.setEnabled(false);  
 deleteRowsOfListOfDoctorsButton.setEnabled(false);  
 throwOffSearchListOfDoctorButton.setEnabled(false);  
 sortInfo.setForeground(new Color(255,255,255));  
  
 propertiesListOfDoctorPanel.add(editListOfDoctorsButton, "split 6, gaptop 10, gapleft 45");  
 propertiesListOfDoctorPanel.add(deleteRowsOfListOfDoctorsButton);  
 propertiesListOfDoctorPanel.add(searchListOfDoctorsButton, "split 2, span 1 2, gapleft 120");  
 propertiesListOfDoctorPanel.add(throwOffSearchListOfDoctorButton);  
 propertiesListOfDoctorPanel.add(sortInfo, "gapleft 60");  
  
 tableOfDoctors.setModel(dtm);  
 tableOfDoctors.setRowSorter(rowSorter); // Для сортировки по столбцам  
 tableOfDoctors.setColumnSelectionAllowed(false);  
 tableOfDoctors.getTableHeader().setReorderingAllowed(false);  
 try { dtm.addData(jdbcConnection); } // Добавление в модель таблицы данных из БД  
 catch (SQLException e) { e.printStackTrace(); }  
 dtm.setColumnStaticWidth(tableOfDoctors); // Установка ширины полей  
 tableOfDoctors.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*); // Для добавления полосы прокрутки влево-вправо  
  
 listOfDoctorsPanel.add(scrollTableOfDoctors, "push, grow, al center");  
 ///////////////////propertiesAndListOfDoctorsPanel(end)/////////////////////////////////////////////////////////////////////////////////////  
  
 ///////////////////propertiesAndEditListOfDoctorsPanel(begin)///////////////////////////////////////////////////////////////////////////////////  
 propertiesAndEditListOfDoctorsPanel.setBackground(new Color(255,255,255));  
 propertiesAndEditListOfDoctorsPanel.setLayout(new MigLayout());  
  
 editorListOfDoctorsPanel.setBackground(new Color(255,255,255));  
 editorListOfDoctorsPanel.setLayout(new MigLayout());  
  
 editPropertiesListOfDoctorPanel.setBackground(new Color(100,100,100));  
 editPropertiesListOfDoctorPanel.setLayout(new MigLayout());  
  
 applyEditListOfDoctorsButton.setEnabled(false);  
  
 propertiesAndEditListOfDoctorsPanel.add(editorListOfDoctorsPanel, "pushx, growx, h 600!, wrap");  
 propertiesAndEditListOfDoctorsPanel.add(editPropertiesListOfDoctorPanel, "growx, pushx, h 60!");  
  
 editPropertiesListOfDoctorPanel.add(applyEditListOfDoctorsButton, "gaptop 10, split 2, al center");  
 editPropertiesListOfDoctorPanel.add(cancelEditListOfDoctorsButton, "gaptop 10");  
  
 editorTableOfDoctors.setModel(edtm);  
 editorTableOfDoctors.setColumnSelectionAllowed(false);  
 editorTableOfDoctors.getTableHeader().setReorderingAllowed(false);  
 edtm.setColumnStaticWidth(editorTableOfDoctors);  
 editorTableOfDoctors.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 editorListOfDoctorsPanel.add(scrollEditorTableOfDoctors, "push, grow, al center");  
 ///////////////////propertiesAndEditListOfDoctorsPanel(end)/////////////////////////////////////////////////////////////////////////////////////  
////////////////listOfDoctorPanel(end)/////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
////////////////timetablePanel(begin)/////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 //////////////propertiesAndTimetablePanel(begin)//////////////////////////////////////////////////////////////////////////////////  
 propertiesAndTimetablePanel.setBackground(new Color(255,255,255));  
 propertiesAndTimetablePanel.setLayout(new MigLayout());  
  
 timetablePanel.setBackground(new Color(255,255,255));  
 timetablePanel.setLayout(new MigLayout());  
  
 propertiesTimetablePanel.setBackground(new Color(100,100,100));  
 propertiesTimetablePanel.setLayout(new MigLayout());  
  
 propertiesAndTimetablePanel.add(timetablePanel, "pushx, growx, h 600!, wrap");  
 propertiesAndTimetablePanel.add(propertiesTimetablePanel, "growx, pushx, h 60!");  
  
 editTimetableButton.setEnabled(false);  
 throwOffSearchTimetableButton.setEnabled(false);  
 previousWeekButton.setEnabled(false);  
  
 propertiesTimetablePanel.add(editTimetableButton, "split 6, gaptop 10, gapleft 45");  
 propertiesTimetablePanel.add(searchTimetableButton, "split 2, span 1 2, gapleft 60");  
 propertiesTimetablePanel.add(throwOffSearchTimetableButton);  
 propertiesTimetablePanel.add(previousWeekButton, "split 2, span 1 2, gapleft 60");  
 propertiesTimetablePanel.add(nextWeekButton);  
  
 this.ttm = new timetableTableModel(tableOfTimetable, dtm.getDataArrayList());  
 tableOfTimetable.setModel(ttm);  
 tableOfTimetable.setColumnSelectionAllowed(false); // Запрет на перестановку столбцов  
 tableOfTimetable.getTableHeader().setReorderingAllowed(false);  
 rowSorterTimetable = new TableRowSorter(ttm);  
 tableOfTimetable.setRowSorter(rowSorterTimetable); // Для сортировки по столбцам  
 try  
 {  
 ttm.updateDate(jdbcConnection); // Добавление дат в ДБ или их обновление  
 ttm.addData(jdbcConnection); // Добавление в модель таблицы данных из БД  
 ttm.updateDateWeek(jdbcConnection); // Смещение дат в БД  
 }  
 catch (SQLException e) { e.printStackTrace(); }  
 ttm.setColumnStaticWidth(tableOfTimetable); // Установка ширины полей  
 tableOfTimetable.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*); // Для добавления полосы прокрутки влево-вправо  
  
 timetablePanel.add(scrollTableOfTimetable, "push, grow, al center");  
 //////////////propertiesAndTimetablePanel(end)////////////////////////////////////////////////////////////////////////////////////  
  
 //////////////propertiesAndEditTimetablePanel(begin)//////////////////////////////////////////////////////////////////////////////////  
 propertiesAndEditTimetablePanel.setBackground(new Color(255,255,255));  
 propertiesAndEditTimetablePanel.setLayout(new MigLayout());  
  
 editPropertiesTimetablePanel.setBackground(new Color(100, 100, 100));  
 editPropertiesTimetablePanel.setLayout(new MigLayout());  
  
 editorTimetablePanel.setBackground(new Color(255,255,255));  
 editorTimetablePanel.setLayout(new MigLayout());  
  
 propertiesAndEditTimetablePanel.add(editorTimetablePanel, "pushx, growx, h 600!, wrap");  
 propertiesAndEditTimetablePanel.add(editPropertiesTimetablePanel, "growx, pushx, h 60!");  
  
 applyEditTimetableButton.setEnabled(false);  
 cancelEditTimetableButton.setEnabled(true);  
  
 editPropertiesTimetablePanel.add(applyEditTimetableButton, "gaptop 10, split 2, al center");  
 editPropertiesTimetablePanel.add(cancelEditTimetableButton, "gaptop 10");  
  
 ettm = new editorTimetableTableModel(this, editorTableOfTimetable, applyEditTimetableButton, ttm.getColumnTypes(), ttm.getColumnNames(), ttm.getColumnRUSNames(), ttm.getDataArrayList());  
 editorTableOfTimetable.setModel(ettm);  
 editorTableOfTimetable.setColumnSelectionAllowed(false);  
 editorTableOfTimetable.getTableHeader().setReorderingAllowed(false);  
 ettm.setColumnStaticWidth(editorTableOfTimetable);  
 editorTableOfTimetable.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
  
 editorTimetablePanel.add(scrollEditorTableOfTimetable, "push, grow, al center");  
 //////////////propertiesAndEditTimetablePanel(end)////////////////////////////////////////////////////////////////////////////////////  
////////////////timetablePanel(end)///////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////////receptionPanel(begin)////////////////////////////////////////////////////////////////////////////////////////  
 receptionPanel.setBackground(new Color(255,255,255));  
 receptionPanel.setLayout(new MigLayout());  
  
 receptionTablePanel.setBackground(new Color(255,255,255));  
 receptionTablePanel.setLayout(new MigLayout());  
  
 propertiesReceptionPanel.setBackground(new Color(100,100,100));  
 propertiesReceptionPanel.setLayout(new MigLayout());  
  
 receptionPanel.add(receptionTablePanel, "pushx, growx, h 500!, wrap");  
 receptionPanel.add(propertiesReceptionPanel, "growx, pushx, h 160!");  
  
 addToReceptionButton.setEnabled(false);  
 cancelFromReceptionButton.setEnabled(false);  
  
 chooseSpecializationLabel.setForeground(new Color(255,255,255));  
 chooseDoctorLabel.setForeground(new Color(255,255,255));  
 chooseDateLabel.setForeground(new Color(255,255,255));  
 showHoursLabel.setForeground(new Color(255,255,255));  
 chooseTimeLabel.setForeground(new Color(255,255,255));  
 choosePatientLabel.setForeground(new Color(255,255,255));  
  
 Vector<String> sp = new Vector();  
 sp.addAll(jdbcConnection.getAllSpecialization());  
 sp.add("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  
 chooseSpecializationJComboBox = new JComboBox(sp);  
 //chooseSpecializationJComboBox.setFocusable(false);  
 chooseSpecializationJComboBox.setSelectedIndex( sp.size() - 1 );  
 //chooseDateJComboBox.setFocusable(false);  
 //chooseTimeJComboBox.setFocusable(false);  
 showHoursText.setEditable(false);  
  
 propertiesReceptionPanel.add(chooseSpecializationLabel, "gaptop 10");  
 propertiesReceptionPanel.add(chooseDoctorLabel, "gaptop 10, gapleft 40");  
 propertiesReceptionPanel.add(chooseDateLabel, "gaptop 10, gapleft 40");  
 propertiesReceptionPanel.add(showHoursLabel, "gaptop 10, gapleft 40");  
 propertiesReceptionPanel.add(chooseTimeLabel, "gaptop 10, gapleft 40");  
 propertiesReceptionPanel.add(choosePatientLabel, "gaptop 10, gapleft 40, wrap");  
 propertiesReceptionPanel.add(chooseSpecializationJComboBox, "gaptop 10");  
 propertiesReceptionPanel.add(scrollChooseDoctorJList, "w 200!, gaptop 10, gapleft 40");  
 propertiesReceptionPanel.add(chooseDateJComboBox, "w 100!, gaptop 10, gapleft 40");  
 propertiesReceptionPanel.add(showHoursText, "w 90!, gaptop 10, gapleft 40");  
 propertiesReceptionPanel.add(chooseTimeJComboBox, "w 80!, gaptop 10, gapleft 40");  
 propertiesReceptionPanel.add(scrollChoosePatientJList, "w 300!, gaptop 10, gapleft 40, wrap");  
 propertiesReceptionPanel.add(addToReceptionButton, "span 2 1, al center");  
 propertiesReceptionPanel.add(cancelFromReceptionButton, "span 2 1, al center");  
  
 this.rtm = new receptionTableModel(tableOfReception);  
 tableOfReception.setModel(rtm);  
 tableOfReception.setRowSorter(rowSorterReception); // Для сортировки по столбцам  
 tableOfReception.setColumnSelectionAllowed(false);  
 tableOfReception.getTableHeader().setReorderingAllowed(false);  
 try { rtm.addData(jdbcConnection); } // Добавление в модель таблицы данных из БД  
 catch (SQLException e) { e.printStackTrace(); }  
 rtm.setColumnStaticWidth(tableOfReception); // Установка ширины полей  
 rowSorterReception = new TableRowSorter(rtm);  
 tableOfReception.setRowSorter(rowSorterReception); // Для сортировки по столбцам  
 tableOfReception.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*); // Для добавления полосы прокрутки влево-вправо  
  
 receptionTablePanel.add(scrollTableOfReception, "push, grow, al center");  
/////////////////////receptionPanel(end)//////////////////////////////////////////////////////////////////////////////////////////  
  
  
/////////////////cardPanel(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 CardLayout cl = new CardLayout();  
 cardPanel.setLayout(cl);  
 cardPanel.add(startRegistryMainAdminWindow, "startPage");  
 cardPanel.add(voidReferenceListOfDoctorsPanel, "voidListOfDoctorsPanel");  
 cardPanel.add(voidReferenceListOfPatientsPanel, "voidListOfPatientsPanel");  
 cardPanel.add(voidReferenceAdminTimetablePanel, "voidTimetablePanel");  
 cardPanel.add(voidServiceAdminReceptionPanel, "voidReceptionPanel");  
 cardPanel.add(referenceRegistryMainAdminWindow, "referencePage");  
 cardPanel.add(serviceRegistryMainAdminWindow, "servicePage");  
 cardPanel.add(newDataRegistryMainAdminWindow, "newDataPage");  
 cardPanel.add(voidNewDataRegistryMainAdminWindow, "voidNewDataPanel");  
 cl.show(cardPanel, "startPage");  
/////////////////cardPanel(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////adminTablePanel(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 CardLayout cl2 = new CardLayout();  
 adminTablePanel.setLayout(cl2);  
 adminTablePanel.add(voidPanel, "voidPanel");  
 adminTablePanel.add(scrollRegisterDoctor, "registerDoctor");  
 adminTablePanel.add(scrollRegisterPatient, "registerPatient");  
 adminTablePanel.add(scrollPropertyAndListOfDoctorsPanel, "listOfDoctorsPanel");  
 adminTablePanel.add(scrollPropertyAndEditListOfDoctorsPanel, "editListOfDoctors");  
 adminTablePanel.add(scrollPropertyAndListOfPatientsPanel, "listOfPatientsPanel");  
 adminTablePanel.add(scrollPropertyAndEditListOfPatientsPanel, "editListOfPatients");  
 adminTablePanel.add(scrollPropertiesAndTimetablePanel, "timetablePanel");  
 adminTablePanel.add(scrollReceptionPanel, "receptionPanel");  
 adminTablePanel.add(scrollPropertyAndEditTimetablePanel, "editTimetable");  
 cl2.show(adminTablePanel, "voidPanel");  
/////////////////adminTablePanel(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
/////////////////registryMainWindowPanels(begin)////////////////////////////////////////////////////////////////////////////////////////////////////////////  
 adminTablePanel.setBackground(new Color(255,255,0));  
 adminTablePanel.setSize(200, (int) screenSize.getHeight());  
 cardPanel.setBackground(new Color(255,0,0));  
 cardPanel.setSize(1166, (int) screenSize.getHeight());  
  
 mainAdminWindowPanel.setLayout(new MigLayout());  
 mainAdminWindowPanel.add(cardPanel, "w 250!, pushy, growy");  
 mainAdminWindowPanel.add(adminTablePanel, "pushx, pushy, growx, growy");  
/////////////////registryMainWindowPanels(end)//////////////////////////////////////////////////////////////////////////////////////////////////////////////  
  
 addRegistryMainWindow(scrollMainAdminWindowPanel, (int)(screenSize.getWidth()), (int)(screenSize.getHeight()));  
  
/////////////ActionListeners(begin)//////////////////////////////////////////////////////////////////////////////////////////////////  
 backToStartFromNewDataRegistryMainAdminWindow.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "startPage");  
 cl2.show(adminTablePanel, "voidPanel");  
 });  
 backToStartFromServiceRegistryMainAdminWindow.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "startPage");  
 });  
 backToStartFromReferenceRegistryMainAdminWindow.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "startPage");  
 cl2.show(adminTablePanel, "voidPanel");  
 });  
 backToReferenceFromVoidListOfPatientsPanelButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "referencePage");  
 cl2.show(adminTablePanel, "voidPanel");  
 tableOfPatients.clearSelection();  
 editListOfPatientsButton.setEnabled(false);  
 deleteRowsOfListOfPatientsButton.setEnabled(false);  
  
 rowSorterPatient.setRowFilter(new myRowFilter("", 10).*regexFilter*("" ));  
 searchListOfPatientsButton.setEnabled(true);  
 throwOffSearchListOfPatientButton.setEnabled(false);  
 });  
 backToReferenceFromVoidListOfDoctorsPanelButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "referencePage");  
 cl2.show(adminTablePanel, "voidPanel");  
 tableOfDoctors.clearSelection();  
 editorTableOfDoctors.clearSelection();  
 editorTableOfDoctors.removeEditor();  
 editListOfDoctorsButton.setEnabled(false);  
 deleteRowsOfListOfDoctorsButton.setEnabled(false);  
  
 rowSorter.setRowFilter(new myRowFilter("", 10).*regexFilter*("" ));  
 searchListOfDoctorsButton.setEnabled(true);  
 throwOffSearchListOfDoctorButton.setEnabled(false);  
 });  
 voidBackToStartFromNewDataRegistryMainAdminWindow.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "newDataPage");  
 cl2.show(adminTablePanel, "voidPanel");  
  
 secondNameDoctorText.setText("");  
 firstNameDoctorText.setText("");  
 thirdNameDoctorText.setText("");  
 maleDoctorCheckBox.setSelected(false);  
 femaleDoctorCheckBox.setSelected(false);  
 birthdayDayDoctorTable.setDate(new Date());  
 numberOfPlotDoctorComboBox.setSelectedIndex(0);  
 specializationOfDoctorComboBox.setSelectedIndex(0);  
 dateOfStartJobDoctorDate.setDate(new Date());  
 newPasswordDoctorText.setText("");  
 mailAddressDoctorText.setText("");  
 cityOfDoctorText.setText("");  
 streetOfDoctorText.setText("");  
 buildingOfDoctorText.setText("");  
 apartmentOfDoctorText.setText("");  
 notDoctorData.setVisible(false);  
 });  
 backToReferenceFromVoidTimetablePanelButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "referencePage");  
 cl2.show(adminTablePanel, "voidPanel");  
 tableOfTimetable.clearSelection();  
 editorTableOfTimetable.removeEditor();  
 ttm.setCurrentStateID(0);  
 editTimetableButton.setEnabled(false);  
 searchTimetableButton.setEnabled(true);  
 throwOffSearchTimetableButton.setEnabled(false);  
 previousWeekButton.setEnabled(false);  
 nextWeekButton.setEnabled(true);  
 rowSorterTimetable.setRowFilter(new myRowFilter("", 10).*regexFilter*("" ));  
 });  
 backToServiceFromVoidReceptionPanelButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "servicePage");  
 cl2.show(adminTablePanel, "voidPanel");  
 tableOfReception.clearSelection();  
 addToReceptionButton.setEnabled(false);  
 cancelFromReceptionButton.setEnabled(false);  
 rowSorterReception.setRowFilter(new myRowFilter("", 9).*regexFilter*("" ));  
  
 chooseSpecializationJComboBox.setSelectedIndex(sp.size() - 1);  
  
 chooseDoctorJListModel.removeAllElements();  
 chooseDoctorJList.setModel(chooseDoctorJListModel);  
  
 chooseDateJComboBoxModel.removeAllElements();  
 chooseDateJComboBox.setModel(chooseDateJComboBoxModel);  
  
 showHoursText.setText("");  
  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
  
 choosePatientJListModel.removeAllElements();  
 choosePatientJList.setModel(choosePatientJListModel);  
 });  
  
 startAdminNewDataButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "newDataPage");  
 });  
 startAdminServiceButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "servicePage");  
 });  
 startAdminReferenceButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "referencePage");  
 });  
  
 serviceAdminDoctorVisitButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "voidReceptionPanel");  
 cl2.show(adminTablePanel, "receptionPanel");  
 });  
  
 newDataAdminAddDoctorButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "voidNewDataPanel");  
 cl2.show(adminTablePanel, "registerDoctor");  
 });  
 newDataAdminAddPatientButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "voidNewDataPanel");  
 cl2.show(adminTablePanel, "registerPatient");  
 });  
  
 registerDoctorButton.addActionListener(actionEvent -> {  
 registerNewDoctor();  
 try { dtm.updateData(jdbcConnection); } // Добавление в модель таблицы данных из БД (обновление списка)  
 catch (SQLException e) { e.printStackTrace(); }  
 try { ttm.updateData(jdbcConnection); } // Добавление в модель таблицы данных из БД (обновление списка)  
 catch (SQLException e) { e.printStackTrace(); }  
 });  
 registerPatientButton.addActionListener(actionEvent -> {  
 registerNewPatient();  
 try { ptm.updateData(jdbcConnection); } // Добавление в модель таблицы данных из БД (обновление списка)  
 catch (SQLException e) { e.printStackTrace(); }  
 });  
  
 referenceAdminListPatientButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "voidListOfPatientsPanel");  
 cl2.show(adminTablePanel, "listOfPatientsPanel");  
 });  
 referenceAdminListDoctorButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "voidListOfDoctorsPanel");  
 cl2.show(adminTablePanel, "listOfDoctorsPanel");  
 });  
 referenceAdminTimetableButton.addActionListener(actionEvent -> {  
 cl.show(cardPanel, "voidTimetablePanel");  
 cl2.show(adminTablePanel, "timetablePanel");  
 });  
  
 maleDoctorCheckBox.addActionListener(actionEvent -> {  
 if(maleDoctorCheckBox.isSelected())  
 {  
 femaleDoctorCheckBox.setSelected(false);  
 }  
 });  
 femaleDoctorCheckBox.addActionListener(actionEvent -> {  
 if(femaleDoctorCheckBox.isSelected())  
 {  
 maleDoctorCheckBox.setSelected(false);  
 }  
 });  
 malePatientCheckBox.addActionListener(actionEvent -> {  
 if(malePatientCheckBox.isSelected())  
 {  
 femalePatientCheckBox.setSelected(false);  
 }  
 });  
 femalePatientCheckBox.addActionListener(actionEvent -> {  
 if(femalePatientCheckBox.isSelected())  
 {  
 malePatientCheckBox.setSelected(false);  
 }  
 });  
  
 editListOfDoctorsButton.addActionListener(actionEvent -> {  
 cl2.show(adminTablePanel, "editListOfDoctors");  
 edtm.addData(tableOfDoctors.getSelectedRows(), tableOfDoctors);  
 tableOfDoctors.clearSelection();  
 editListOfDoctorsButton.setEnabled(false);  
 deleteRowsOfListOfDoctorsButton.setEnabled(false);  
 });  
 deleteRowsOfListOfDoctorsButton.addActionListener(actionEvent -> {  
 ttm.deleteData( dtm.deleteData(tableOfDoctors.getSelectedRows(), tableOfDoctors, jdbcConnection)); // Удаление сначала произойдёт в dtm, потом в ttm  
 tableOfDoctors.clearSelection();  
 editListOfDoctorsButton.setEnabled(false);  
 deleteRowsOfListOfDoctorsButton.setEnabled(false);  
 });  
 applyEditListOfDoctorsButton.addActionListener(actionEvent -> {  
 edtm.editData();  
 applyEditListOfDoctorsButton.setEnabled(false);  
 try { dtm.addData(jdbcConnection); } // Добавление в модель таблицы данных из БД  
 catch (SQLException e) { e.printStackTrace(); }  
 try { ttm.addData(jdbcConnection); } // Добавление в модель таблицы данных из БД  
 catch (SQLException e) { e.printStackTrace(); }  
 });  
 cancelEditListOfDoctorsButton.addActionListener(actionEvent -> {  
 cl2.show(adminTablePanel, "listOfDoctorsPanel");  
 applyEditListOfDoctorsButton.setEnabled(false);  
 editListOfDoctorsButton.setEnabled(false);  
 deleteRowsOfListOfDoctorsButton.setEnabled(false);  
 tableOfDoctors.clearSelection();  
 editorTableOfDoctors.removeEditor();  
 });  
 searchListOfDoctorsButton.addActionListener(actionEvent -> {  
 new searchDoctors(this, throwOffSearchListOfDoctorButton, searchListOfDoctorsButton, rowSorter, dtm);  
 });  
 throwOffSearchListOfDoctorButton.addActionListener(actionEvent -> {  
 rowSorter.setRowFilter(new myRowFilter("", 10).*regexFilter*("" ));  
 searchListOfDoctorsButton.setEnabled(true);  
 throwOffSearchListOfDoctorButton.setEnabled(false);  
 tableOfDoctors.clearSelection();  
 editListOfDoctorsButton.setEnabled(false);  
 deleteRowsOfListOfDoctorsButton.setEnabled(false);  
 });  
  
 searchTimetableButton.addActionListener(actionEvent -> {  
 new searchTimetable(this, throwOffSearchTimetableButton, searchTimetableButton, rowSorterTimetable, ttm);  
 });  
 throwOffSearchTimetableButton.addActionListener(actionEvent -> {  
 rowSorterTimetable.setRowFilter(new myRowFilter("", 10).*regexFilter*("" )); // Пока rowFilter общий  
 searchTimetableButton.setEnabled(true);  
 throwOffSearchTimetableButton.setEnabled(false);  
 tableOfTimetable.clearSelection();  
 editTimetableButton.setEnabled(false);  
 });  
 previousWeekButton.addActionListener(actionEvent -> {  
 ttm.setCurrentStateID(ttm.getCurrentStateID() - 1);  
 ettm.setColumnRUSNames(ttm.getColumnRUSNames());  
  
 if(ttm.getCurrentStateID() == 0)  
 {  
 previousWeekButton.setEnabled(false);  
 nextWeekButton.setEnabled(true);  
 ettm.setCurrentStateID(ttm.getCurrentStateID());  
 }  
 else if(ttm.getCurrentStateID() == 1)  
 {  
 previousWeekButton.setEnabled(true);  
 nextWeekButton.setEnabled(true);  
 ettm.setCurrentStateID(ttm.getCurrentStateID());  
 }  
 else if(ttm.getCurrentStateID() == 2)  
 {  
 previousWeekButton.setEnabled(true);  
 nextWeekButton.setEnabled(false);  
 ettm.setCurrentStateID(ttm.getCurrentStateID());  
 }  
 });  
 nextWeekButton.addActionListener(actionEvent -> {  
 ttm.setCurrentStateID(ttm.getCurrentStateID() + 1);  
 ettm.setColumnRUSNames(ttm.getColumnRUSNames());  
  
 if(ttm.getCurrentStateID() == 0)  
 {  
 previousWeekButton.setEnabled(false);  
 nextWeekButton.setEnabled(true);  
 ettm.setCurrentStateID(ttm.getCurrentStateID());  
 }  
 else if(ttm.getCurrentStateID() == 1)  
 {  
 previousWeekButton.setEnabled(true);  
 nextWeekButton.setEnabled(true);  
 ettm.setCurrentStateID(ttm.getCurrentStateID());  
 }  
 else if(ttm.getCurrentStateID() == 2)  
 {  
 previousWeekButton.setEnabled(true);  
 nextWeekButton.setEnabled(false);  
 ettm.setCurrentStateID(ttm.getCurrentStateID());  
 }  
 });  
 editTimetableButton.addActionListener(actionEvent -> {  
 cl2.show(adminTablePanel, "editTimetable");  
 ettm.addData(tableOfTimetable.getSelectedRows(), tableOfTimetable);  
 tableOfTimetable.clearSelection();  
 editTimetableButton.setEnabled(false);  
 });  
 applyEditTimetableButton.addActionListener(actionEvent -> {  
 ettm.editData();  
 applyEditTimetableButton.setEnabled(false);  
 try { ttm.addData(jdbcConnection); } // Добавление в модель таблицы данных из БД  
 catch (SQLException e) { e.printStackTrace(); }  
 });  
 cancelEditTimetableButton.addActionListener(actionEvent -> {  
 cl2.show(adminTablePanel, "timetablePanel");  
 editorTableOfTimetable.removeEditor();  
 ttm.setCurrentStateID(0);  
 nextWeekButton.setEnabled(true);  
 previousWeekButton.setEnabled(false);  
 });  
  
 searchListOfPatientsButton.addActionListener(actionEvent -> {  
 new searchPatients(this, throwOffSearchListOfPatientButton, searchListOfPatientsButton, rowSorterPatient, ptm, tableOfPatients);  
 });  
 throwOffSearchListOfPatientButton.addActionListener(actionEvent -> {  
 rowSorterPatient.setRowFilter(new myRowFilter("", 14).*regexFilter*("" ));  
 searchListOfPatientsButton.setEnabled(true);  
 throwOffSearchListOfPatientButton.setEnabled(false);  
 tableOfPatients.clearSelection();  
 editListOfPatientsButton.setEnabled(false);  
 deleteRowsOfListOfPatientsButton.setEnabled(false);  
 });  
 deleteRowsOfListOfPatientsButton.addActionListener(actionEvent -> {  
 ptm.deleteData(tableOfPatients.getSelectedRows(), tableOfPatients, jdbcConnection); // Удаление сначала произойдёт в dtm, потом в ttm  
 tableOfPatients.clearSelection();  
 editListOfPatientsButton.setEnabled(false);  
 deleteRowsOfListOfPatientsButton.setEnabled(false);  
 });  
 editListOfPatientsButton.addActionListener(actionEvent -> {  
 cl2.show(adminTablePanel, "editListOfPatients");  
 eptm.addData(tableOfPatients.getSelectedRows(), tableOfPatients);  
 tableOfPatients.clearSelection();  
 editListOfPatientsButton.setEnabled(false);  
 deleteRowsOfListOfPatientsButton.setEnabled(false);  
 });  
 applyEditListOfPatientsButton.addActionListener(actionEvent -> {  
 eptm.editData();  
 applyEditListOfPatientsButton.setEnabled(false);  
 try { ptm.addData(jdbcConnection); } // Добавление в модель таблицы данных из БД  
 catch (SQLException e) { e.printStackTrace(); }  
 });  
 cancelEditListOfPatientsButton.addActionListener(actionEvent -> {  
 cl2.show(adminTablePanel, "listOfPatientsPanel");  
 applyEditListOfPatientsButton.setEnabled(false);  
 editListOfPatientsButton.setEnabled(false);  
 deleteRowsOfListOfPatientsButton.setEnabled(false);  
 tableOfPatients.clearSelection();  
 editorTableOfPatients.removeEditor();  
 });  
  
 chooseSpecializationJComboBox.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e)  
 {  
 Vector<String> temp = new Vector();  
 if(e.getStateChange() != ItemEvent.*DESELECTED*)  
 {  
 for (int i = 0; i < dtm.getDataArrayList().size(); i++) {  
 if (((ArrayList) dtm.getDataArrayList().get(i)).get(6).equals(e.getItem().toString())) {  
 temp.add(((ArrayList) dtm.getDataArrayList().get(i)).get(1).toString() + " " +  
 ((ArrayList) dtm.getDataArrayList().get(i)).get(2).toString() + " " +  
 ((ArrayList) dtm.getDataArrayList().get(i)).get(3).toString());  
 }  
 }  
 if (!temp.isEmpty()) {  
 chooseDoctorJListModel.removeAllElements();  
 chooseDoctorJListModel.addAll(temp /\*jdbcConnection.getNeedDoctors( e.getItem().toString() )\*/);  
 chooseDoctorJList.setModel(chooseDoctorJListModel);  
 showHoursText.setText("");  
 chooseDateJComboBoxModel.removeAllElements();  
 chooseDateJComboBox.setModel(chooseDateJComboBoxModel);  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 if(!choosePatientJListModel.isEmpty())  
 {  
 choosePatientJListModel.removeAllElements();  
 choosePatientJList.setModel(chooseDoctorJListModel);  
 }  
 } else {  
 chooseDoctorJListModel.removeAllElements();  
 chooseDoctorJList.setModel(chooseDoctorJListModel);  
 showHoursText.removeAll();  
 }  
 }  
 else  
 {  
 for (int i = 0; i < dtm.getDataArrayList().size(); i++) {  
 if (((ArrayList) dtm.getDataArrayList().get(i)).get(6).equals(e.getItem().toString())) {  
 temp.add(((ArrayList) dtm.getDataArrayList().get(i)).get(1).toString() + " " +  
 ((ArrayList) dtm.getDataArrayList().get(i)).get(2).toString() + " " +  
 ((ArrayList) dtm.getDataArrayList().get(i)).get(3).toString());  
 }  
 }  
 chooseDoctorJListModel.removeAllElements();  
 chooseDoctorJListModel.addAll(temp /\*jdbcConnection.getNeedDoctors( e.getItem().toString() )\*/);  
 chooseDoctorJList.setModel(chooseDoctorJListModel);  
 showHoursText.setText("");  
 chooseDateJComboBoxModel.removeAllElements();  
 chooseDateJComboBox.setModel(chooseDateJComboBoxModel);  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 if(!choosePatientJListModel.isEmpty())  
 {  
 choosePatientJListModel.removeAllElements();  
 choosePatientJList.setModel(chooseDoctorJListModel);  
 }  
 addToReceptionButton.setEnabled(false);  
 }  
 }  
 });  
 chooseDoctorJList.addListSelectionListener(new ListSelectionListener() {  
 @Override  
 public void valueChanged(ListSelectionEvent e)  
 {  
 Vector<String> temp = new Vector();  
 if( e.getValueIsAdjusting() && !chooseDoctorJList.isSelectionEmpty() )  
 {  
 for (int i = 0; i < ttm.getCurrentState().size(); i++)  
 {  
 temp.add(ttm.getWeeks(i).getPnDate().toString());  
 temp.add(ttm.getWeeks(i).getVtDate().toString());  
 temp.add(ttm.getWeeks(i).getSrDate().toString());  
 temp.add(ttm.getWeeks(i).getChtDate().toString());  
 temp.add(ttm.getWeeks(i).getPtDate().toString());  
 temp.add(ttm.getWeeks(i).getSbDate().toString());  
 }  
 showHoursText.setText("");  
 chooseDateJComboBoxModel.removeAllElements();  
 chooseDateJComboBox.setModel(chooseDateJComboBoxModel);  
 chooseDateJComboBoxModel.addAll(temp /\*jdbcConnection.getNeedDoctors( e.getItem().toString() )\*/);  
 chooseDateJComboBox.setModel(chooseDateJComboBoxModel);  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 if(!choosePatientJListModel.isEmpty()) {  
 choosePatientJListModel.removeAllElements();  
 choosePatientJList.setModel(chooseDoctorJListModel);  
 }  
 addToReceptionButton.setEnabled(false);  
 }  
 else if( chooseDoctorJList.isSelectionEmpty() )  
 {  
 chooseDateJComboBoxModel.removeAllElements();  
 chooseDateJComboBox.setModel(chooseDateJComboBoxModel);  
 showHoursText.removeAll();  
 }  
 }  
 });  
 chooseDateJComboBox.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e)  
 {  
 if (e.getStateChange() != ItemEvent.*DESELECTED*)  
 {  
 String temp = "Не принимает";  
 boolean be = false;  
 for (int i = 0; i < ttm.getDataArrayList().size(); i++) {  
 if (((ArrayList) ttm.getDataArrayList().get(i)).get(0).toString().equals(chooseDoctorJList.getSelectedValue())) {  
 for (int j = 3; j < 21; j++) {  
 String[] edit = new Scanner(((ArrayList) ttm.getDataArrayList().get(i)).get(j).toString()).nextLine().split(" ");  
 String chooseDate = chooseDateJComboBox.getSelectedItem().toString().replace(" ", "");  
 if (chooseDate.equals(edit[edit.length - 3]) && !edit[2].equals("--:--/--:--")) {  
 temp = edit[2];  
 be = true;  
 break;  
 }  
 }  
 }  
 if (be) break;  
 }  
  
 if (chooseDateJComboBoxModel.getSize() != 0 && !temp.isEmpty()) {  
 showHoursText.removeAll();  
 showHoursText.setText(temp);  
  
 Vector<String> temp2 = new Vector();  
 if (!temp.equals("Не принимает")) {  
 temp2 = addTimeCorrectly(temp2);  
  
 if (!temp2.isEmpty()) {  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBoxModel.addAll(temp2);  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 } else {  
 temp2.add("Нет времени");  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBoxModel.addAll(temp2);  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 choosePatientJListModel.removeAllElements();  
 choosePatientJList.setModel(choosePatientJListModel);  
 addToReceptionButton.setEnabled(false);  
 }  
 } else {  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBoxModel.addElement("Нет времени");  
 choosePatientJListModel.removeAllElements();  
 choosePatientJList.setModel(choosePatientJListModel);  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 addToReceptionButton.setEnabled(false);  
 }  
 } else {  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 addToReceptionButton.setEnabled(false);  
 showHoursText.removeAll();  
 }  
 }  
 else  
 {  
  
 }  
 }  
 });  
 chooseTimeJComboBox.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e)  
 {  
 if(chooseTimeJComboBoxModel.getSize() != 0)  
 {  
 choosePatientJListModel.removeAllElements();  
 choosePatientJListModel.addAll( jdbcConnection.getNeedPatients() );  
 choosePatientJList.setModel(choosePatientJListModel);  
 }  
 else  
 {  
 addToReceptionButton.setEnabled(false);  
 choosePatientJListModel.removeAllElements();  
 choosePatientJList.setModel(choosePatientJListModel);  
 showHoursText.removeAll();  
 }  
 }  
 });  
 choosePatientJList.addListSelectionListener(new ListSelectionListener() {  
 @Override  
 public void valueChanged(ListSelectionEvent e)  
 {  
 if( e.getValueIsAdjusting() && !choosePatientJList.isSelectionEmpty() )  
 {  
 addToReceptionButton.setEnabled(true);  
 }  
 else if(choosePatientJList.isSelectionEmpty())  
 {  
  
 }  
 else if( chooseTimeJComboBoxModel.getSize() == 0 || chooseTimeJComboBoxModel.getElementAt(0).equals("Нет времени") )  
 {  
 choosePatientJListModel.removeAllElements();  
 choosePatientJList.setModel(choosePatientJListModel);  
 showHoursText.removeAll();  
 addToReceptionButton.setEnabled(false);  
 }  
 }  
 });  
 addToReceptionButton.addActionListener(actionEvent -> {  
 rtm.writeInBD(dtm.getDataArrayList(), chooseDoctorJList.getSelectedValue(), choosePatientJList.getSelectedValue(), chooseDateJComboBox.getSelectedItem().toString(), chooseTimeJComboBox.getSelectedItem().toString());  
 tableOfReception.clearSelection();  
 try { rtm.addData(jdbcConnection); }  
 catch (SQLException e) { e.printStackTrace(); }  
 rtm.fireTableDataChanged();  
 addToReceptionButton.setEnabled(false);  
 cancelFromReceptionButton.setEnabled(false);  
  
 chooseTimeJComboBoxModel.removeElementAt( chooseTimeJComboBox.getSelectedIndex() );  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 });  
 cancelFromReceptionButton.addActionListener(actionEvent -> {  
 rtm.deleteData(tableOfReception.getSelectedRows(), tableOfReception, jdbcConnection);  
  
 if(chooseDateJComboBoxModel.getSize() != 0)  
 {  
 Vector<String> temp2 = new Vector();  
 temp2 = addTimeCorrectly(temp2);  
 chooseTimeJComboBoxModel.removeAllElements();  
 chooseTimeJComboBoxModel.addAll(temp2);  
 chooseTimeJComboBox.setModel(chooseTimeJComboBoxModel);  
 }  
  
 tableOfReception.clearSelection();  
 cancelFromReceptionButton.setEnabled(false);  
 });  
  
 tableOfTimetable.setSelectionModel(new DefaultListSelectionModel() // Модель выбора для удобного интерфейса  
 {  
 @Override  
 public void setSelectionInterval(int startIndex, int endIndex) {  
 if (startIndex == endIndex) // Если выбран один элемент  
 {  
 if (getMinSelectionIndex() != getMaxSelectionIndex())  
 {  
 clearSelection(); // Убрать выделение  
 editTimetableButton.setEnabled(false);  
 }  
 if (isSelectedIndex(startIndex))  
 {  
 clearSelection(); // Убрать выделение  
 editTimetableButton.setEnabled(false);  
 }  
 else  
 {  
 super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
 editTimetableButton.setEnabled(true);  
 }  
 }  
 // Если выбраны многие элементы  
 else {  
 super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
 editTimetableButton.setEnabled(true);  
 }  
 }  
 });  
 tableOfDoctors.setSelectionModel(new DefaultListSelectionModel() // Модель выбора для удобного интерфейса  
 {  
 @Override  
 public void setSelectionInterval(int startIndex, int endIndex) {  
 if (startIndex == endIndex) // Если выбран один элемент  
 {  
 if (getMinSelectionIndex() != getMaxSelectionIndex())  
 {  
 clearSelection(); // Убрать выделение  
 editListOfDoctorsButton.setEnabled(false);  
 deleteRowsOfListOfDoctorsButton.setEnabled(false);  
 }  
 if (isSelectedIndex(startIndex))  
 {  
 clearSelection(); // Убрать выделение  
 editListOfDoctorsButton.setEnabled(false);  
 deleteRowsOfListOfDoctorsButton.setEnabled(false);  
 }  
 else  
 {  
 super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
 editListOfDoctorsButton.setEnabled(true);  
 deleteRowsOfListOfDoctorsButton.setEnabled(true);  
 }  
 }  
 // Если выбраны многие элементы  
 else {  
 super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
 editListOfDoctorsButton.setEnabled(true);  
 deleteRowsOfListOfDoctorsButton.setEnabled(true);  
 }  
 }  
 });  
 tableOfPatients.setSelectionModel(new DefaultListSelectionModel() // Модель выбора для удобного интерфейса  
 {  
 @Override  
 public void setSelectionInterval(int startIndex, int endIndex) {  
 if (startIndex == endIndex) // Если выбран один элемент  
 {  
 if (getMinSelectionIndex() != getMaxSelectionIndex())  
 {  
 clearSelection(); // Убрать выделение  
 editListOfPatientsButton.setEnabled(false);  
 deleteRowsOfListOfPatientsButton.setEnabled(false);  
 }  
 if (isSelectedIndex(startIndex))  
 {  
 clearSelection(); // Убрать выделение  
 editListOfPatientsButton.setEnabled(false);  
 deleteRowsOfListOfPatientsButton.setEnabled(false);  
 }  
 else  
 {  
 super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
 editListOfPatientsButton.setEnabled(true);  
 deleteRowsOfListOfPatientsButton.setEnabled(true);  
 }  
 }  
 // Если выбраны многие элементы  
 else {  
 super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
 editListOfPatientsButton.setEnabled(true);  
 deleteRowsOfListOfPatientsButton.setEnabled(true);  
 }  
 }  
 });  
 tableOfReception.setSelectionModel(new DefaultListSelectionModel() // Модель выбора для удобного интерфейса  
 {  
 @Override  
 public void setSelectionInterval(int startIndex, int endIndex) {  
 if (startIndex == endIndex) // Если выбран один элемент  
 {  
 if (getMinSelectionIndex() != getMaxSelectionIndex())  
 {  
 clearSelection(); // Убрать выделение  
 cancelFromReceptionButton.setEnabled(false);  
 }  
 if (isSelectedIndex(startIndex))  
 {  
 clearSelection(); // Убрать выделение  
 cancelFromReceptionButton.setEnabled(false);  
 }  
 else  
 {  
 super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
 cancelFromReceptionButton.setEnabled(true);  
 }  
 }  
 // Если выбраны многие элементы  
 else {  
 super.setSelectionInterval(startIndex, endIndex); // Добавить выделение  
 cancelFromReceptionButton.setEnabled(true);  
 }  
 }  
 });  
/////////////ActionListeners(end)////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 private Vector<String> addTimeCorrectly(Vector<String> temp2)  
 {  
 Statement statement = null;  
 ResultSet resSet = null;  
 try  
 {  
 for (int i = 0; i < ttm.getDataArrayList().size(); i++)  
 {  
 if (((ArrayList) ttm.getDataArrayList().get(i)).get(0).toString().equals(chooseDoctorJList.getSelectedValue()))  
 {  
 statement = jdbcConnection.getDbConnecion().createStatement();  
 resSet = jdbcConnection.getReceptionForCheck((Integer)((ArrayList) ttm.getDataArrayList().get(i)).get(21), chooseDateJComboBox.getSelectedItem().toString(), statement);  
  
 String[] edit = new Scanner(showHoursText.getText()).nextLine().split("/");  
 String[] editBegin = new Scanner(edit[0]).nextLine().split(":");  
 String[] editEnd = new Scanner(edit[1]).nextLine().split(":");  
 int b1 = Integer.*parseInt*(editBegin[0]); int b2 = Integer.*parseInt*(editBegin[1]);  
 int e1 = Integer.*parseInt*(editEnd[0]); int e2 = Integer.*parseInt*(editEnd[1]);  
  
 int minutes = b2;  
 for(int hours = b1; hours <= e1; hours++)  
 {  
 if(hours > b1) minutes = 0;  
 for(; minutes < 60; minutes += 10)  
 {  
 if(hours == e1)  
 {  
 if(minutes <= e2)  
 {  
 if (hours - 10 >= 0 && minutes > 0) temp2.add(String.*valueOf*(hours) + ":" + String.*valueOf*(minutes));  
 else if (hours - 10 >= 0 && minutes == 0) temp2.add(String.*valueOf*(hours) + ":" + String.*valueOf*(minutes) + "0");  
 else if (hours - 10 < 0 && minutes > 0) temp2.add("0" + String.*valueOf*(hours) + ":" + String.*valueOf*(minutes));  
 else if (hours - 10 < 0 && minutes == 0) temp2.add("0" + String.*valueOf*(hours) + ":" + String.*valueOf*(minutes) + "0");  
 }  
 }  
 else  
 {  
 if (hours - 10 >= 0 && minutes > 0) temp2.add(String.*valueOf*(hours) + ":" + String.*valueOf*(minutes));  
 else if (hours - 10 >= 0 && minutes == 0) temp2.add(String.*valueOf*(hours) + ":" + String.*valueOf*(minutes) + "0");  
 else if (hours - 10 < 0 && minutes > 0) temp2.add("0" + String.*valueOf*(hours) + ":" + String.*valueOf*(minutes));  
 else if (hours - 10 < 0 && minutes == 0) temp2.add("0" + String.*valueOf*(hours) + ":" + String.*valueOf*(minutes) + "0");  
 }  
 }  
 }  
  
 while(resSet.next())  
 {  
 for (int ii = 0; ii < temp2.size(); ii++)  
 {  
 if (resSet.getString(CONST.*RECEPTION\_TIME\_OF\_RECEPTION*).equals(temp2.get(ii)))  
 {  
 temp2.removeElementAt(ii);  
 break;  
 }  
 }  
 }  
  
 if(!resSet.next())  
 {  
 break;  
 }  
 }  
 }  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 } catch (ClassNotFoundException ex) {  
 ex.printStackTrace();  
 }  
 finally  
 {  
 try {  
 jdbcConnection.dbDisconnectionSt(statement);  
 jdbcConnection.dbDisconnectionResSet(resSet);  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 return temp2;  
 }  
  
 private void registerNewPatient()  
 {  
 if(!secondNamePatientText.getText().equals("") && !firstNamePatientText.getText().equals("") &&  
 !thirdNamePatientText.getText().equals("") && (malePatientCheckBox.isSelected() || femalePatientCheckBox.isSelected()) &&  
 dateOfCreatePatientCard.getDate().before(new Date()) && !numberOfInsurancePolicyText.getText().equals("") &&  
 !numberOfPhonePatientText.getText().equals("") && !numberOfPassportPatientText.getText().equals("") &&  
 !cityOfPatientText.getText().equals("") && !streetOfPatientText.isSelectionEmpty() &&  
 !buildingOfPatientText.getText().equals("") && !apartmentOfPatientText.getText().equals("") &&  
 birthdayDayPatientTable.getDate().before(new Date())) // Если все регистрационные поля заполнены  
 {  
 notPatientData.setVisible(false);  
  
 java.sql.Date date = new java.sql.Date(birthdayDayPatientTable.getDate().getTime());  
 java.sql.Date date2 = new java.sql.Date(dateOfCreatePatientCard.getDate().getTime());  
  
 String male\_feemale = "";  
 if (malePatientCheckBox.isSelected()) male\_feemale = "Мужской";  
 else male\_feemale = "Женский";  
  
 Patient patient = new Patient(secondNamePatientText.getText().trim(), firstNamePatientText.getText().trim(), thirdNamePatientText.getText().trim(),  
 male\_feemale, date, date2, numberOfPassportPatientText.getText().trim(), numberOfPhonePatientText.getText().trim(),  
 numberOfInsurancePolicyText.getText().trim(), cityOfPatientText.getText().trim(), streetOfPatientText.getSelectedValue(),  
 buildingOfPatientText.getText().trim(), apartmentOfPatientText.getText().trim());  
  
 jdbcConnection.registerPatient(patient);  
  
 secondNamePatientText.setText("");  
 firstNamePatientText.setText("");  
 thirdNamePatientText.setText("");  
 malePatientCheckBox.setSelected(false);  
 femalePatientCheckBox.setSelected(false);  
 dateOfCreatePatientCard.setDate(new Date());  
 numberOfInsurancePolicyText.setText("");  
 numberOfPhonePatientText.setText("");  
 numberOfPassportPatientText.setText("");  
 cityOfPatientText.setText("");  
 streetOfPatientText.setSelectedIndex(0);  
 buildingOfPatientText.setText("");  
 apartmentOfPatientText.setText("");  
 birthdayDayPatientTable.setDate(new Date());  
  
 new infoMessage(this, "Новый пациент зарегистрирован в системе", "УВЕДОМЛЕНИЕ");  
 }  
 else  
 {  
 notPatientData.setVisible(true);  
 }  
 }  
  
 private void registerNewDoctor()  
 {  
 if(!secondNameDoctorText.getText().equals("") && !firstNameDoctorText.getText().equals("") &&  
 !thirdNameDoctorText.getText().equals("") && (maleDoctorCheckBox.isSelected() || femaleDoctorCheckBox.isSelected()) &&  
 dateOfStartJobDoctorDate.getDate().before(new Date()) &&  
 !newPasswordDoctorText.getText().equals("") && !mailAddressDoctorText.getText().equals("") &&  
 !cityOfDoctorText.getText().equals("") && !streetOfDoctorText.getText().equals("") &&  
 !buildingOfDoctorText.getText().equals("") && !apartmentOfDoctorText.getText().equals("") &&  
 birthdayDayDoctorTable.getDate().before(new Date())) // Если все регистрационные поля заполнены  
 {  
 notDoctorData.setVisible(false);  
  
 java.sql.Date date = new java.sql.Date(birthdayDayDoctorTable.getDate().getTime());  
  
 java.sql.Date date2 = new java.sql.Date(dateOfStartJobDoctorDate.getDate().getTime());  
  
 String male\_feemale = "";  
 if (maleDoctorCheckBox.isSelected()) male\_feemale = "Мужской";  
 else male\_feemale = "Женский";  
  
 Doctor doctor = new Doctor(secondNameDoctorText.getText().trim(), firstNameDoctorText.getText().trim(),  
 thirdNameDoctorText.getText().trim(), male\_feemale, date,  
 specializationOfDoctorComboBox.getItemAt(specializationOfDoctorComboBox.getSelectedIndex()),  
 numberOfPlotDoctorComboBox.getItemAt(numberOfPlotDoctorComboBox.getSelectedIndex()),  
 date2, newPasswordDoctorText.getText(), mailAddressDoctorText.getText().trim(),  
 cityOfDoctorText.getText().trim(), streetOfDoctorText.getText().trim(),  
 buildingOfDoctorText.getText().trim(), apartmentOfDoctorText.getText().trim());  
  
 jdbcConnection.registerDoctor(doctor);  
  
 secondNameDoctorText.setText("");  
 firstNameDoctorText.setText("");  
 thirdNameDoctorText.setText("");  
 maleDoctorCheckBox.setSelected(false);  
 femaleDoctorCheckBox.setSelected(false);  
 birthdayDayDoctorTable.setDate(new Date());  
 numberOfPlotDoctorComboBox.setSelectedIndex(0);  
 specializationOfDoctorComboBox.setSelectedIndex(0);  
 dateOfStartJobDoctorDate.setDate(new Date());  
 newPasswordDoctorText.setText("");  
 mailAddressDoctorText.setText("");  
 cityOfDoctorText.setText("");  
 streetOfDoctorText.setText("");  
 buildingOfDoctorText.setText("");  
 apartmentOfDoctorText.setText("");  
 notDoctorData.setVisible(false);  
  
 new infoMessage(this, "Новый врач зарегистрирован в системе", "УВЕДОМЛЕНИЕ");  
 }  
 else  
 {  
 notDoctorData.setVisible(true);  
 }  
 }  
  
 private void addRegistryMainWindow(JScrollPane \_panel, int width, int height)  
 {  
 this.setBounds(0, 0, width, height); // Устанавливаем оптимальные размеры окна, учитывая размеры экрана  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*); // При нажатии на X произойдёт выход из приложения  
 this.setFocusable(true); // Фокус на окне  
 this.setVisible(true); // Делаем окно видимым  
 this.setResizable(true); // Запрет на изменение размеров окна - нет  
 this.setExtendedState(this.*MAXIMIZED\_BOTH*); // Растяжение на весь экран  
 this.setTitle("Регистратура"); // Заголовок окна  
  
 this.add(\_panel);  
 }  
  
 private void refreshRegistryMainWindow(int width, int height)  
 {  
 this.setBounds(0, 0, width, height); // Устанавливаем оптимальные размеры окна, учитывая размеры экрана  
 this.setLocationRelativeTo(null); // Расположение окна ровно по центру  
 }  
}

* КЛАСС JDBCConnection
* package Connector;  
    
  import Users.Admin;  
  import Users.Doctor;  
  import Users.Patient;  
    
  import javax.swing.\*;  
  import javax.swing.plaf.nimbus.State;  
  import javax.xml.transform.Result;  
  import java.math.BigInteger;  
  import java.sql.\*;  
  import java.sql.ResultSet;  
  import java.util.ArrayList;  
  import java.util.Vector;  
  import java.util.regex.Matcher;  
  import java.util.regex.Pattern;  
    
  public class JDBCConnection extends Configs  
  {  
   private Connection dbConnecion;  
    
   public Connection getDbConnecion() throws ClassNotFoundException, SQLException  
   {  
   Class.*forName*("com.mysql.jdbc.Driver"); // Загрузка класса драйвера, создание мостика  
   String sourceURL = "jdbc:mysql://" + dbHost + ":" + dbPort + "/" + dbName; // Источник данных  
   dbConnecion = DriverManager.*getConnection*(sourceURL, dbUser, dbPassword); // Установка соединения с БД  
    
   return dbConnecion;  
   }  
    
   public void dbDisconnectionPrSt(PreparedStatement prSt) throws SQLException  
   {  
   if(prSt != null)  
   {  
   prSt.close();  
   }  
   }  
   public void dbDisconnectionResSet(ResultSet resSet) throws SQLException  
   {  
   if(resSet != null)  
   {  
   resSet.close();  
   }  
   }  
   public void dbDisconnectionSt(Statement st) throws SQLException  
   {  
   if(st != null)  
   {  
   st.close();  
   }  
   }  
   public void dbDisconnectionCs(CallableStatement cs) throws SQLException  
   {  
   if(cs != null)  
   {  
   cs.close();  
   }  
   }  
    
   // Методы записи/чтения в/из базы данных  
   public void registerAdmin(Admin admin) // Регистрация администратора в БД  
   {  
   String insert = "INSERT INTO " + CONST.*ADMIN\_TABLE* + "(" // SQL запрос: в таблицу admins, в следующие поля вставить следующие данные  
   + CONST.*ADMINS\_SECONDNAME* + "," + CONST.*ADMINS\_FIRSTNAME* + ","  
   + CONST.*ADMINS\_THIRDNAME* + "," + CONST.*ADMINS\_GENDER* + ","  
   + CONST.*ADMINS\_BIRTHDAY* + "," + CONST.*ADMINS\_PASSWORD* + ","  
   + CONST.*ADMINS\_MAIL* + "," + CONST.*ADMINS\_CITY* + ","  
   + CONST.*ADMINS\_STREET* + "," + CONST.*ADMINS\_BUILDING* + ","  
   + CONST.*ADMINS\_APARTMENT* + ")" + "VALUES(?,?,?,?,?,?,?,?,?,?,?)";  
    
   PreparedStatement prSt = null;  
    
   try  
   {  
   prSt = getDbConnecion().prepareStatement(insert); // PreparedStatement - служит для выполнения параметризованных запросов в виде ?  
    
   prSt.setString(1, admin.getSecondName());  
   prSt.setString(2, admin.getFirstName());  
   prSt.setString(3, admin.getThirdName());  
   prSt.setString(4, admin.getGender());  
   prSt.setDate(5, admin.getBirthday());  
   prSt.setString(6, admin.getPassword());  
   prSt.setString(7, admin.getMailAddress());  
   prSt.setString(8, admin.getCity());  
   prSt.setString(9, admin.getStreet());  
   prSt.setString(10, admin.getBuilding());  
   prSt.setString(11, admin.getApartment());  
    
   prSt.executeUpdate(); // Выполнить SQL запрос о передаче данных в БД  
   }  
   catch (ClassNotFoundException e) // Если не найдена БД (Не загрузился Driver)  
   {  
   System.*out*.println(e);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
   finally // Обязательное закрытие подключения к БД  
   {  
   try  
   {  
   dbDisconnectionPrSt(prSt);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
   }  
    
   public Doctor getDoctor(Doctor doctor) // Возвращает массив данных об администраторе из БД  
   {  
   PreparedStatement prSt = null;  
    
   String select = "SELECT \* FROM " + CONST.*DOCTOR\_TABLE* + " doc INNER JOIN " + CONST.*SPECIALIZATION\_TABLE* + " sp INNER JOIN " + CONST.*PLOT\_TABLE* + " pl" +  
   " ON " + "doc." + CONST.*DOCTORS\_SPECIALIZATION\_CODE* + " = sp." + CONST.*SPECIALIZATION\_CODE* + " AND doc." + CONST.*DOCTORS\_TABLE\_NUMBER* + " = pl." + CONST.*PLOTS\_DOCTORS\_TABLE\_NUMBER* +  
   " WHERE " + CONST.*DOCTORS\_MAIL* + "=? AND " + CONST.*DOCTORS\_PASSWORD* + "=?"; // SQL запрос: из таблицы admins вытянуть все данные: почта и пароль, которые равны следующим значениям  
    
   ResultSet resSet = null;  
    
   try  
   {  
   prSt = getDbConnecion().prepareStatement(select); // PreparedStatement - служит для выполнения параметризованных запросов в виде ?  
    
   prSt.setString(1, doctor.getMailAddress());  
   prSt.setString(2, doctor.getPassword());  
    
   resSet = prSt.executeQuery(); // Выполнить SQL запрос о взятии данных из БД (и записать результат в переменную)  
    
   try  
   {  
   while(resSet.next())  
   {  
   doctor.setSecondName(resSet.getString(CONST.*DOCTORS\_SECONDNAME*));  
   doctor.setFirstName(resSet.getString(CONST.*DOCTORS\_FIRSTNAME*));  
   doctor.setThirdName(resSet.getString(CONST.*DOCTORS\_THIRDNAME*));  
   doctor.setMailAddress(resSet.getString(CONST.*DOCTORS\_MAIL*));  
   doctor.setPassword(resSet.getString(CONST.*DOCTORS\_PASSWORD*));  
   doctor.setBirthday(resSet.getDate(CONST.*DOCTORS\_BIRTHDAY*));  
   doctor.setGender(resSet.getString(CONST.*DOCTORS\_GENDER*));  
   doctor.setCity(resSet.getString(CONST.*DOCTORS\_CITY*));  
   doctor.setStreet(resSet.getString(CONST.*DOCTORS\_STREET*));  
   doctor.setBuilding(resSet.getString(CONST.*DOCTORS\_BUILDING*));  
   doctor.setApartment(resSet.getString(CONST.*DOCTORS\_APARTMENT*));  
   doctor.setDateOfStartJob(resSet.getDate(CONST.*DOCTORS\_DATE\_START\_JOB*));  
   doctor.setSpecialization(resSet.getString(CONST.*SPECIALIZATION*));  
   doctor.setNumberOfPlot(resSet.getInt(CONST.*PLOTS\_NUMBER*));  
   doctor.setTableNumber(resSet.getInt(CONST.*DOCTORS\_TABLE\_NUMBER*));  
   }  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
   catch (ClassNotFoundException e) // Если не найдена БД  
   {  
   System.*out*.println(e);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД  
   {  
   System.*out*.println(e);  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionPrSt(prSt);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
    
   return doctor; // Если администратор не будет найден, то вернётся null в каждом поле  
   }  
    
   public Admin getAdmin(Admin admin) // Возвращает массив данных об администраторе из БД  
   {  
   PreparedStatement prSt = null;  
    
   String select = "SELECT \* FROM " + CONST.*ADMIN\_TABLE* + " WHERE "  
   + CONST.*ADMINS\_MAIL* + "=? AND " + CONST.*ADMINS\_PASSWORD* + "=?"; // SQL запрос: из таблицы admins вытянуть все данные: почта и пароль, которые равны следующим значениям  
    
   ResultSet resSet = null;  
    
   try  
   {  
   prSt = getDbConnecion().prepareStatement(select); // PreparedStatement - служит для выполнения параметризованных запросов в виде ?  
    
   prSt.setString(1, admin.getMailAddress());  
   prSt.setString(2, admin.getPassword());  
    
   resSet = prSt.executeQuery(); // Выполнить SQL запрос о взятии данных из БД (и записать результат в переменную)  
    
   try  
   {  
   while(resSet.next())  
   {  
   admin.setSecondName(resSet.getString(CONST.*ADMINS\_SECONDNAME*));  
   admin.setFirstName(resSet.getString(CONST.*ADMINS\_FIRSTNAME*));  
   admin.setThirdName(resSet.getString(CONST.*ADMINS\_THIRDNAME*));  
   admin.setMailAddress(resSet.getString(CONST.*ADMINS\_MAIL*));  
   admin.setPassword(resSet.getString(CONST.*ADMINS\_PASSWORD*));  
   admin.setBirthday(resSet.getDate(CONST.*ADMINS\_BIRTHDAY*));  
   admin.setGender(resSet.getString(CONST.*ADMINS\_GENDER*));  
   admin.setCity(resSet.getString(CONST.*ADMINS\_CITY*));  
   admin.setStreet(resSet.getString(CONST.*ADMINS\_STREET*));  
   admin.setBuilding(resSet.getString(CONST.*ADMINS\_BUILDING*));  
   admin.setApartment(resSet.getString(CONST.*ADMINS\_APARTMENT*));  
   }  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
   catch (ClassNotFoundException e) // Если не найдена БД  
   {  
   System.*out*.println(e);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД  
   {  
   System.*out*.println(e);  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionPrSt(prSt);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
    
   return admin; // Если администратор не будет найден, то вернётся null в каждом поле  
   }  
    
   public void registerDoctor(Doctor doctor)  
   {  
   String insert = "INSERT INTO " + CONST.*DOCTOR\_TABLE* + "(" // SQL запрос: в таблицу doctors, в следующие поля вставить следующие данные  
   + CONST.*DOCTORS\_SECONDNAME* + "," + CONST.*DOCTORS\_FIRSTNAME* + ","  
   + CONST.*DOCTORS\_THIRDNAME* + "," + CONST.*DOCTORS\_GENDER* + ","  
   + CONST.*DOCTORS\_BIRTHDAY* + "," + CONST.*DOCTORS\_SPECIALIZATION\_CODE* + ","  
   + CONST.*DOCTORS\_DATE\_START\_JOB* + "," + CONST.*DOCTORS\_PASSWORD* + ","  
   + CONST.*DOCTORS\_MAIL* + "," + CONST.*DOCTORS\_CITY* + ","  
   + CONST.*DOCTORS\_STREET* + "," + CONST.*DOCTORS\_BUILDING* + "," + CONST.*DOCTORS\_APARTMENT* + ")" + "VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)";  
    
   String select = "SELECT \* FROM " + CONST.*SPECIALIZATION\_TABLE* + " WHERE " + CONST.*SPECIALIZATION* + "=?";  
    
   String insert2 = "INSERT INTO " + CONST.*PLOT\_TABLE* + "(" + CONST.*PLOTS\_DOCTORS\_TABLE\_NUMBER* + ","  
   + CONST.*PLOTS\_NUMBER* + ")" + " VALUES(?,?)";  
    
   String select2 = "SELECT \* FROM " + CONST.*DOCTOR\_TABLE* + " WHERE "  
   + CONST.*DOCTORS\_SECONDNAME* + "=? AND " + CONST.*DOCTORS\_FIRSTNAME* + "=? AND "  
   + CONST.*DOCTORS\_THIRDNAME* + "=? AND " + CONST.*DOCTORS\_GENDER* + "=? AND "  
   + CONST.*DOCTORS\_BIRTHDAY* + "=? AND "  
   + CONST.*DOCTORS\_DATE\_START\_JOB* + "=? AND " + CONST.*DOCTORS\_PASSWORD* + "=? AND "  
   + CONST.*DOCTORS\_MAIL* + "=? AND " + CONST.*DOCTORS\_CITY* + "=? AND "  
   + CONST.*DOCTORS\_STREET* + "=? AND " + CONST.*DOCTORS\_BUILDING* + "=? AND " + CONST.*DOCTORS\_APARTMENT* + "=?";  
    
   String insert3 = "INSERT INTO " + CONST.*TIMETABLE\_TABLE* + "(" + CONST.*TIMETABLE\_TABLE\_NUMBER* + ","  
   + CONST.*TIMETABLE\_NUMBER\_CABINET* + ")" + " VALUES(?,?)";  
    
   String insert4 = "INSERT INTO " + CONST.*TIMETABLETIME\_TABLE* + "(" + CONST.*TIMETABLETIME\_TABLE\_NUMBER* + "," +  
   CONST.*TIMETABLETIME\_ID\_DAY* + ")" + " VALUES(?,?)";  
    
   String insert5 = "INSERT INTO " + CONST.*TIMETABLEDAY\_TABLE* + "(" + CONST.*TIMETABLEDAY\_TABLE\_NUMBER* + "," +  
   CONST.*TIMETABLEDAY\_ID\_DAY* + ")" + " VALUES(?,?)";  
    
   String insert6 = "INSERT INTO " + CONST.*TIMEOFJOB\_TABLE* + "(" + CONST.*TIMEOFJOB\_TABLE\_NUMBER* + "," +  
   CONST.*TIMEOFJOB\_ID\_TIME\_BEGIN* + "," + CONST.*TIMEOFJOB\_ID\_TIME\_END* + "," + CONST.*TIMEOFJOB\_ID\_WEEK* + ")" + " VALUES(?,?,?,?)";  
    
   PreparedStatement prStIns = null;  
   PreparedStatement prStSel = null;  
   PreparedStatement prStIns2 = null;  
   PreparedStatement prStSel2 = null;  
   PreparedStatement prStIns3 = null;  
   PreparedStatement prStIns4 = null;  
   PreparedStatement prStIns5 = null;  
   PreparedStatement prStIns6 = null;  
   ResultSet resSet = null;  
   ResultSet resSet2 = null;  
    
   try  
   {  
   prStIns = getDbConnecion().prepareStatement(insert); // PreparedStatement - служит для выполнения параметризованных запросов в виде ?  
   prStSel = getDbConnecion().prepareStatement(select);  
   prStIns2 = getDbConnecion().prepareStatement(insert2);  
   prStSel2 = getDbConnecion().prepareStatement(select2);  
   prStIns3 = getDbConnecion().prepareStatement(insert3);  
   prStIns4 = getDbConnecion().prepareStatement(insert4);  
   prStIns5 = getDbConnecion().prepareStatement(insert5);  
   prStIns6 = getDbConnecion().prepareStatement(insert6);  
    
   prStSel.setString(1, doctor.getSpecialization());  
   resSet = prStSel.executeQuery();  
    
   prStIns.setString(1, doctor.getSecondName());  
   prStIns.setString(2, doctor.getFirstName());  
   prStIns.setString(3, doctor.getThirdName());  
   prStIns.setString(4, doctor.getGender());  
   prStIns.setDate(5, doctor.getBirthday());  
   while(resSet.next())  
   {  
   prStIns.setInt(6, resSet.getInt(CONST.*SPECIALIZATION\_CODE*));  
   }  
   prStIns.setDate(7, doctor.getDateOfStartJob());  
   prStIns.setString(8, doctor.getPassword());  
   prStIns.setString(9, doctor.getMailAddress());  
   prStIns.setString(10, doctor.getCity());  
   prStIns.setString(11, doctor.getStreet());  
   prStIns.setString(12, doctor.getBuilding());  
   prStIns.setString(13, doctor.getApartment());  
    
   prStIns.executeUpdate(); // Выполнить SQL запрос о передаче данных в БД  
    
    
    
    
   prStSel2.setString(1, doctor.getSecondName());  
   prStSel2.setString(2, doctor.getFirstName());  
   prStSel2.setString(3, doctor.getThirdName());  
   prStSel2.setString(4, doctor.getGender());  
   prStSel2.setDate(5, doctor.getBirthday());  
   prStSel2.setDate(6, doctor.getDateOfStartJob());  
   prStSel2.setString(7, doctor.getPassword());  
   prStSel2.setString(8, doctor.getMailAddress());  
   prStSel2.setString(9, doctor.getCity());  
   prStSel2.setString(10, doctor.getStreet());  
   prStSel2.setString(11, doctor.getBuilding());  
   prStSel2.setString(12, doctor.getApartment());  
   resSet2 = prStSel2.executeQuery();  
    
   int tableNumber = 0;  
   while(resSet2.next())  
   {  
   //prStIns2.setInt(1, resSet2.getInt(CONST.DOCTORS\_TABLE\_NUMBER));  
   tableNumber = resSet2.getInt(CONST.*DOCTORS\_TABLE\_NUMBER*);  
   }  
   prStIns2.setInt(1, tableNumber);  
   prStIns2.setInt(2, doctor.getNumberOfPlot());  
   prStIns2.executeUpdate();  
    
    
    
    
   prStIns3.setInt(1, tableNumber);  
   prStIns3.setInt(2, 0);  
   prStIns3.executeUpdate();  
    
    
    
   for(int i = 1; i <= 21; i++)  
   {  
   if(i <= 6)  
   {  
   prStIns4.setInt(1, tableNumber);  
   prStIns4.setInt(2, i);  
    
   prStIns5.setInt(1, tableNumber);  
   prStIns5.setInt(2, i);  
    
   prStIns6.setInt(1, tableNumber);  
   prStIns6.setInt(2, 23);  
   prStIns6.setInt(3, 23);  
   prStIns6.setInt(4, i);  
    
    
   prStIns4.executeUpdate();  
   prStIns5.executeUpdate();  
   prStIns6.executeUpdate();  
   }  
   else if(i > 6 && i <= 21)  
   {  
   prStIns6.setInt(1, tableNumber);  
   prStIns6.setInt(2, 23);  
   prStIns6.setInt(3, 23);  
   prStIns6.setInt(4, i);  
    
   prStIns6.executeUpdate();  
   }  
   }  
   }  
   catch (ClassNotFoundException e) // Если не найдена БД (Не загрузился Driver)  
   {  
   System.*out*.println(e);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
   finally // Обязательное закрытие подключения к БД  
   {  
   try  
   {  
   dbDisconnectionPrSt(prStIns);  
   dbDisconnectionPrSt(prStSel);  
   dbDisconnectionPrSt(prStIns2);  
   dbDisconnectionPrSt(prStSel2);  
   dbDisconnectionResSet(resSet);  
   dbDisconnectionResSet(resSet2);  
   dbDisconnectionPrSt(prStIns3);  
   dbDisconnectionPrSt(prStIns4);  
   dbDisconnectionPrSt(prStIns5);  
   dbDisconnectionPrSt(prStIns6);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
   }  
    
   public Vector<String> getAllStrees()  
   {  
   Vector<String> streets = new Vector();  
   String select = "SELECT " + CONST.*ADDRESSPLOT\_STREET* + " FROM " + CONST.*ADDRESSPLOT\_TABLE*;  
   ResultSet resSet = null;  
   Statement st = null;  
    
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select);  
   while(resSet.next())  
   {  
   streets.add(resSet.getString(CONST.*ADDRESSPLOT\_STREET*));  
   }  
   }  
   catch (SQLException e) { System.*out*.println(e); } catch (ClassNotFoundException e) { e.printStackTrace(); }  
   finally  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e) { e.printStackTrace(); }  
   }  
    
   return streets;  
   }  
    
   public ArrayList getStreetsAndPlots()  
   {  
   ArrayList streets = new ArrayList();  
    
   String select = "SELECT " + CONST.*ADDRESSPLOT\_NUMBER\_OF\_PLOT* + "," + CONST.*ADDRESSPLOT\_STREET* + " FROM " + CONST.*ADDRESSPLOT\_TABLE*;  
   ResultSet resSet = null;  
   Statement st = null;  
    
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select);  
   while(resSet.next())  
   {  
   ArrayList temp = new ArrayList();  
   temp.add(resSet.getInt(CONST.*ADDRESSPLOT\_NUMBER\_OF\_PLOT*));  
   temp.add(resSet.getString(CONST.*ADDRESSPLOT\_STREET*));  
    
   streets.add(temp);  
   }  
   }  
   catch (SQLException e) { System.*out*.println(e); } catch (ClassNotFoundException e) { e.printStackTrace(); }  
   finally  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e) { e.printStackTrace(); }  
   }  
    
   return streets;  
   }  
    
   public void registerPatient(Patient patient)  
   {  
   String insert = "INSERT INTO " + CONST.*PATIENT\_TABLE* + "("  
   + CONST.*PATIENT\_SECONDNAME* + "," + CONST.*PATIENT\_FIRSTNAME* + ","  
   + CONST.*PATIENT\_THIRDNAME* + "," + CONST.*PATIENT\_GENDER* + ","  
   + CONST.*PATIENT\_BIRTHDAY* + "," + CONST.*PATIENT\_NUMBER\_OF\_PLOT* + ","  
   + CONST.*PATIENT\_DATE\_OF\_CREATE\_CARD* + "," + CONST.*PATIENT\_NUMBER\_OF\_CARD* + ","  
   + CONST.*PATIENT\_NUMBER\_OF\_PASPORT* + "," + CONST.*PATIENT\_NUMBER\_OF\_PHONE* + ","  
   + CONST.*PATIENT\_INSURANCE\_POLICY* + "," + CONST.*PATIENT\_CITY* + "," + CONST.*PATIENT\_STREET* + ","  
   + CONST.*PATIENT\_BUILDING* + "," + CONST.*PATIENT\_APARTMENT* + ")" + " VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";  
    
   String select = "SELECT " + CONST.*ADDRESSPLOT\_NUMBER\_OF\_PLOT* + "," + CONST.*ADDRESSPLOT\_STREET* + " FROM " + CONST.*ADDRESSPLOT\_TABLE*;  
    
   String select2 = "";  
    
   String update = "UPDATE " + CONST.*PATIENT\_TABLE* + " SET " + CONST.*PATIENT\_NUMBER\_OF\_CARD* + " = ?" + " WHERE " +  
   CONST.*PATIENT\_ID* + " = ?";  
    
   PreparedStatement prStIns = null;  
   PreparedStatement prStUpd = null;  
   Statement stSel = null;  
   Statement stSel2 = null;  
   ResultSet resSet2 = null;  
   ResultSet resSet = null;  
    
   try  
   {  
   prStIns = getDbConnecion().prepareStatement(insert);  
   prStUpd = getDbConnecion().prepareStatement(update);  
   stSel = getDbConnecion().createStatement();  
   stSel2 = getDbConnecion().createStatement();  
    
   resSet = stSel.executeQuery(select);  
    
   int numberOfPlot = 1;  
   while(resSet.next())  
   {  
   if( resSet.getString(CONST.*ADDRESSPLOT\_STREET*).equals(patient.getStreet()) )  
   {  
   numberOfPlot = resSet.getInt(CONST.*ADDRESSPLOT\_NUMBER\_OF\_PLOT*);  
   break;  
   }  
   }  
    
   select2 = "SELECT \* FROM " + CONST.*PATIENT\_TABLE* + " WHERE "  
   + CONST.*PATIENT\_SECONDNAME* + " = '" + patient.getSecondName() + "' AND "  
   + CONST.*PATIENT\_FIRSTNAME* + " = '" + patient.getFirstName() + "' AND "  
   + CONST.*PATIENT\_THIRDNAME* + " = '" + patient.getThirdName() + "' AND "  
   + CONST.*PATIENT\_GENDER* + " = '" + patient.getGender() + "' AND "  
   + CONST.*PATIENT\_BIRTHDAY* + " = DATE '" + patient.getBirthday() + "' AND "  
   + CONST.*PATIENT\_NUMBER\_OF\_PLOT* + " = " + String.*valueOf*(numberOfPlot) + " AND "  
   + CONST.*PATIENT\_DATE\_OF\_CREATE\_CARD* + " = DATE '" + patient.getDayOfCreateCard() + "' AND "  
   + CONST.*PATIENT\_NUMBER\_OF\_PASPORT* + " = '" + patient.getNumberOfPassport() + "' AND "  
   + CONST.*PATIENT\_NUMBER\_OF\_PHONE* + " = '" + patient.getNumberOfPhone() + "' AND "  
   + CONST.*PATIENT\_INSURANCE\_POLICY* + " = '" + patient.getNumberOfInsurancePolicy() + "' AND "  
   + CONST.*PATIENT\_CITY* + " = '" + patient.getCity() + "' AND "  
   + CONST.*PATIENT\_STREET* + " = '" + patient.getStreet() + "' AND "  
   + CONST.*PATIENT\_BUILDING* + " = '" + patient.getBuilding() + "' AND "  
   + CONST.*PATIENT\_APARTMENT* + " = '" + patient.getApartment() + "'";  
    
   prStIns.setString(1, patient.getSecondName());  
   prStIns.setString(2, patient.getFirstName());  
   prStIns.setString(3, patient.getThirdName());  
   prStIns.setString(4, patient.getGender());  
   prStIns.setDate(5, patient.getBirthday());  
   prStIns.setInt(6, numberOfPlot);  
   prStIns.setDate(7, patient.getDayOfCreateCard());  
   try  
   {  
   prStIns.setInt(8, Integer.*parseInt*(patient.getNumberOfPassport()));  
   }catch (NumberFormatException e)  
   {  
   prStIns.setInt(8, (int)(Math.*random*()\*(999999-100000) + 100000) );  
   }  
    
   prStIns.setString(9, patient.getNumberOfPassport());  
   prStIns.setString(10, patient.getNumberOfPhone());  
   prStIns.setString(11, patient.getNumberOfInsurancePolicy());  
   prStIns.setString(12, patient.getCity());  
   prStIns.setString(13, patient.getStreet());  
   prStIns.setString(14, patient.getBuilding());  
   prStIns.setString(15, patient.getApartment());  
    
   prStIns.executeUpdate();  
    
    
   resSet2 = stSel2.executeQuery(select2);  
   while(resSet2.next())  
   {  
   if( resSet2.getString(CONST.*PATIENT\_SECONDNAME*).equals(patient.getSecondName()) &&  
   resSet2.getString(CONST.*PATIENT\_FIRSTNAME*).equals(patient.getFirstName()) &&  
   resSet2.getString(CONST.*PATIENT\_THIRDNAME*).equals(patient.getThirdName()) &&  
   resSet2.getString(CONST.*PATIENT\_GENDER*).equals(patient.getGender()) &&  
   //resSet2.getDate(CONST.PATIENT\_BIRTHDAY).getTime() == patient.getBirthday().getTime() &&  
   resSet2.getInt(CONST.*PATIENT\_NUMBER\_OF\_PLOT*) == numberOfPlot &&  
   //resSet2.getDate(CONST.PATIENT\_DATE\_OF\_CREATE\_CARD).getTime() == patient.getDayOfCreateCard().getTime() &&  
   resSet2.getString(CONST.*PATIENT\_NUMBER\_OF\_PASPORT*).equals(patient.getNumberOfPassport()) &&  
   resSet2.getString(CONST.*PATIENT\_NUMBER\_OF\_PHONE*).equals(patient.getNumberOfPhone()) &&  
   resSet2.getString(CONST.*PATIENT\_INSURANCE\_POLICY*).equals(patient.getNumberOfInsurancePolicy()) &&  
   resSet2.getString(CONST.*PATIENT\_CITY*).equals(patient.getCity()) &&  
   resSet2.getString(CONST.*PATIENT\_STREET*).equals(patient.getStreet()) &&  
   resSet2.getString(CONST.*PATIENT\_BUILDING*).equals(patient.getBuilding()) &&  
   resSet2.getString(CONST.*PATIENT\_APARTMENT*).equals(patient.getApartment()) )  
   {  
   BigInteger local = new BigInteger( String.*valueOf*(resSet2.getInt(CONST.*PATIENT\_ID*)) +  
   resSet2.getDate(CONST.*PATIENT\_BIRTHDAY*).toString().substring(0,4) +  
   resSet2.getDate(CONST.*PATIENT\_BIRTHDAY*).toString().substring(5,7) +  
   resSet2.getDate(CONST.*PATIENT\_BIRTHDAY*).toString().substring(8,10) );  
   prStUpd.setString(1, local.toString() ); // Передаю на сервер текст, который знает, как обработать BIGINT  
   prStUpd.setInt(2, resSet2.getInt(CONST.*PATIENT\_ID*));  
   prStUpd.executeUpdate();  
   break;  
   }  
   }  
   }  
   catch (ClassNotFoundException e) // Если не найдена БД (Не загрузился Driver)  
   {  
   System.*out*.println(e);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
   finally // Обязательное закрытие подключения к БД  
   {  
   try  
   {  
   dbDisconnectionPrSt(prStIns);  
   dbDisconnectionPrSt(prStUpd);  
   dbDisconnectionResSet(resSet);  
   dbDisconnectionResSet(resSet2);  
   dbDisconnectionSt(stSel);  
   dbDisconnectionSt(stSel2);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
   }  
    
   public Vector<String> getNeedPatients()  
   {  
   Vector<String> needPatients = new Vector();  
    
   String select = "SELECT CONCAT(" + CONST.*PATIENT\_NUMBER\_OF\_CARD* + ", ', ', " + CONST.*PATIENT\_SECONDNAME* + ", ' ', " +  
   CONST.*PATIENT\_FIRSTNAME* + ", ' ', " + CONST.*PATIENT\_THIRDNAME* + ") FROM " + CONST.*PATIENT\_TABLE*;  
    
   Statement st = null;  
   ResultSet resSet = null;  
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select); // Выполнить SQL запрос о взятии данных из БД (и записать результат в переменную)  
    
   while(resSet.next())  
   {  
   needPatients.add(resSet.getString(1));  
   }  
   } catch (SQLException e) {  
   e.printStackTrace();  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
    
   return needPatients;  
   }  
    
   public ArrayList getAllSpecialization()  
   {  
   ArrayList specializationList = new ArrayList();  
    
   String select = "SELECT " + CONST.*SPECIALIZATION* + " FROM " + CONST.*SPECIALIZATION\_TABLE*; // SQL запрос: из таблицы admins вытянуть все данные: почта и пароль, которые равны следующим значениям  
    
   Statement st = null;  
   try {  
   st = getDbConnecion().createStatement();  
   } catch (SQLException e) {  
   e.printStackTrace();  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
    
   ResultSet resSet = null;  
   try {  
   resSet = st.executeQuery(select); // Выполнить SQL запрос о взятии данных из БД (и записать результат в переменную)  
   } catch (SQLException e) {  
   e.printStackTrace();  
   }  
    
   try  
   {  
   while(resSet.next())  
   {  
   specializationList.add(resSet.getString(CONST.*SPECIALIZATION*));  
   }  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
    
   return specializationList;  
   }  
    
   public ArrayList getAllSpecializationCode()  
   {  
   ArrayList specializationCodeList = new ArrayList();  
    
   String select = "SELECT " + CONST.*SPECIALIZATION\_CODE* + " FROM " + CONST.*SPECIALIZATION\_TABLE*; // SQL запрос: из таблицы admins вытянуть все данные: почта и пароль, которые равны следующим значениям  
    
   Statement st = null;  
   try {  
   st = getDbConnecion().createStatement();  
   } catch (SQLException e) {  
   e.printStackTrace();  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
    
   ResultSet resSet = null;  
   try {  
   resSet = st.executeQuery(select); // Выполнить SQL запрос о взятии данных из БД (и записать результат в переменную)  
   } catch (SQLException e) {  
   e.printStackTrace();  
   }  
    
   try  
   {  
   while(resSet.next())  
   {  
   specializationCodeList.add(resSet.getInt(CONST.*SPECIALIZATION\_CODE*));  
   }  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
    
   return specializationCodeList;  
   }  
    
   public ArrayList getAllSpecializationANDSpecializationCode()  
   {  
   ArrayList specializationCodeList = new ArrayList();  
    
   String select = "SELECT \* FROM " + CONST.*SPECIALIZATION\_TABLE* + " ORDER BY " + CONST.*SPECIALIZATION\_CODE*; // SQL запрос: из таблицы admins вытянуть все данные: почта и пароль, которые равны следующим значениям  
    
   Statement st = null;  
   try {  
   st = getDbConnecion().createStatement();  
   } catch (SQLException e) {  
   e.printStackTrace();  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
    
   ResultSet resSet = null;  
   try {  
   resSet = st.executeQuery(select); // Выполнить SQL запрос о взятии данных из БД (и записать результат в переменную)  
   } catch (SQLException e) {  
   e.printStackTrace();  
   }  
    
   try  
   {  
   while(resSet.next())  
   {  
   ArrayList temp = new ArrayList();  
   temp.add(resSet.getInt(CONST.*SPECIALIZATION\_CODE*));  
   temp.add(resSet.getString(CONST.*SPECIALIZATION*));  
   specializationCodeList.add(temp);  
   }  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
    
   return specializationCodeList;  
   }  
    
   public Vector<Integer> getCabinetsNumbers()  
   {  
   Vector<Integer> cabinets = new Vector();  
   String select = "SELECT " + CONST.*CABINETS\_NUMBER* + " FROM " + CONST.*CABINETS\_TABLE*;  
   ResultSet resSet = null;  
   Statement st = null;  
    
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select);  
   while(resSet.next())  
   {  
   cabinets.add(resSet.getInt(CONST.*CABINETS\_NUMBER*));  
   }  
   }  
   catch (SQLException e) { System.*out*.println(e); } catch (ClassNotFoundException e) { e.printStackTrace(); }  
   finally  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e) { e.printStackTrace(); }  
   }  
    
   return cabinets;  
   }  
    
   public void writeReception(int tableNumber, String numberOfCard, java.sql.Date date, String time)  
   {  
   String insert = "INSERT INTO " + CONST.*RECEPTION\_TABLE* + "(" +  
   CONST.*RECEPTION\_TABLE\_NUMBER* + ", " +  
   CONST.*RECEPTION\_NUMBER\_OF\_CARD* + ", " +  
   CONST.*RECEPTION\_DATE\_OF\_RECEPTION* + ", " +  
   CONST.*RECEPTION\_TIME\_OF\_RECEPTION* + ", " +  
   CONST.*RECEPTION\_IS\_VISIT* + ")" +  
   " VALUES(" + tableNumber + ", " + numberOfCard + ", DATE '" + date.toString() + "', '" + time + "', false)";  
    
   try  
   {  
   Statement statement = getDbConnecion().createStatement();  
   statement.executeUpdate(insert);  
   } catch (SQLException e) { e.printStackTrace(); } catch (ClassNotFoundException e) { e.printStackTrace(); }  
   }  
    
   public String getComplaint(int[] selectedRows, JTable table)  
   {  
   String complaint = "";  
    
   String select =  
   "SELECT " + CONST.*COMPLAINTS\_COMPLAINTS* + " FROM " + CONST.*COMPLAINTS\_TABLE* +  
   " WHERE " + CONST.*COMPLAINTS\_NUMBER\_OF\_CARD* + " = " + table.getValueAt(selectedRows[0], 2) +  
   " AND " + CONST.*COMPLAINTS\_DATE\_OF\_RECEPTION* + " = DATE '" + table.getValueAt(selectedRows[0], 0) + "'" +  
   " AND " + CONST.*COMPLAINTS\_TIME\_OF\_RECEPTION* + " = '" + table.getValueAt(selectedRows[0], 1) + "'";  
    
   ResultSet resSet = null;  
   Statement st = null;  
    
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select);  
    
   while(resSet.next())  
   {  
   complaint = resSet.getString(CONST.*COMPLAINTS\_COMPLAINTS*);  
   }  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
    
   return complaint;  
   }  
    
   public String getRecomendation(int[] selectedRows, JTable table)  
   {  
   String recomendation = "";  
    
   String select =  
   "SELECT recom." + CONST.*RECOMENDATION\_RECOMENDATION* +  
   " FROM " + CONST.*RECOMENDATION\_TABLE* + " recom INNER JOIN " + CONST.*COMPLAINTS\_TABLE* + " com" +  
   " ON recom." + CONST.*RECOMENDATION\_RECOMENDATION\_CODE* + " = com." + CONST.*COMPLAINTS\_RECOMENDATION\_CODE* +  
   " WHERE " + CONST.*COMPLAINTS\_NUMBER\_OF\_CARD* + " = " + table.getValueAt(selectedRows[0], 2) +  
   " AND " + CONST.*COMPLAINTS\_DATE\_OF\_RECEPTION* + " = DATE '" + table.getValueAt(selectedRows[0], 0) + "'" +  
   " AND " + CONST.*COMPLAINTS\_TIME\_OF\_RECEPTION* + " = '" + table.getValueAt(selectedRows[0], 1) + "'";  
    
   ResultSet resSet = null;  
   Statement st = null;  
    
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select);  
    
   while(resSet.next())  
   {  
   recomendation = resSet.getString(CONST.*RECOMENDATION\_RECOMENDATION*);  
   }  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
    
   return recomendation;  
   }  
    
   public void writeComplaintsANDRecomendationInBD(String complaint, String recomendation, String numberOfCard, String doctorFIO, String date, String time, int tableNumber)  
   {  
   String insert = "INSERT INTO " + CONST.*COMPLAINTS\_TABLE* + "(" + CONST.*COMPLAINTS\_COMPLAINTS* + ", " + CONST.*COMPLAINTS\_NUMBER\_OF\_CARD* + ", " + CONST.*COMPLAINTS\_DATE\_OF\_RECEPTION* + ", " + CONST.*COMPLAINTS\_TIME\_OF\_RECEPTION* + ")" +  
   " VALUES ('" + complaint + "', '" + numberOfCard + "', DATE '" + date + "', '" + time + "')";  
   String insert2 = "INSERT INTO " + CONST.*RECOMENDATION\_TABLE* + "(" + CONST.*RECOMENDATION\_RECOMENDATION* + ", " + CONST.*RECOMENDATION\_RECOMENDATION\_CODE* + ", " + CONST.*RECOMENDATION\_DOCTORSFIO* + ")" +  
   " VALUES ('" + recomendation + "', ?, '" + doctorFIO + "')";  
   String select = "SELECT \* FROM " + CONST.*COMPLAINTS\_TABLE* + " WHERE " + CONST.*COMPLAINTS\_NUMBER\_OF\_CARD* + " = '" + numberOfCard +  
   "' AND " + CONST.*COMPLAINTS\_COMPLAINTS* + " = '" + complaint + "'";  
   String update = "UPDATE " + CONST.*RECEPTION\_TABLE* + " SET " + CONST.*RECEPTION\_IS\_VISIT* + " = true" +  
   " WHERE " + CONST.*RECEPTION\_TABLE\_NUMBER* + " = " + String.*valueOf*(tableNumber) + " AND " + CONST.*RECEPTION\_NUMBER\_OF\_CARD* + " = '" + numberOfCard +  
   "' AND " + CONST.*RECEPTION\_DATE\_OF\_RECEPTION* + " = DATE '" + date + "' AND " + CONST.*RECEPTION\_TIME\_OF\_RECEPTION* + " = '" + time + "'";  
    
   PreparedStatement prSt = null;  
   Statement st = null;  
   Statement st2 = null;  
   Statement st3 = null;  
   ResultSet resSet = null;  
   try  
   {  
   st = getDbConnecion().createStatement();  
   st2 = getDbConnecion().createStatement();  
   st3 = getDbConnecion().createStatement();  
   prSt = getDbConnecion().prepareStatement(insert2);  
    
   st.executeUpdate(insert);  
    
   resSet = st2.executeQuery(select);  
   while(resSet.next())  
   {  
   prSt.setInt(1, resSet.getInt(CONST.*COMPLAINTS\_RECOMENDATION\_CODE*));  
   }  
   prSt.executeUpdate();  
    
   st3.executeUpdate(update);  
   } catch (SQLException e) { e.printStackTrace(); } catch (ClassNotFoundException e) { e.printStackTrace(); }  
   finally  
   {  
   try {  
   dbDisconnectionResSet(resSet);  
   dbDisconnectionSt(st);  
   dbDisconnectionSt(st2);  
   dbDisconnectionSt(st3);  
   dbDisconnectionPrSt(prSt);  
   } catch (SQLException e) {  
   e.printStackTrace();  
   }  
   }  
   }  
    
   public int getComplaintsID(String date, String time, String numberOfCard)  
   {  
   int complaintID = 0;  
    
   String select =  
   "SELECT \* FROM " + CONST.*COMPLAINTS\_TABLE* +  
   " WHERE " + CONST.*COMPLAINTS\_NUMBER\_OF\_CARD* + " = " + numberOfCard +  
   " AND " + CONST.*COMPLAINTS\_DATE\_OF\_RECEPTION* + " = DATE '" + date + "'" +  
   " AND " + CONST.*COMPLAINTS\_TIME\_OF\_RECEPTION* + " = '" + time + "'";  
    
   ResultSet resSet = null;  
   Statement st = null;  
    
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select);  
    
   while(resSet.next())  
   {  
   complaintID = resSet.getInt(CONST.*COMPLAINTS\_RECOMENDATION\_CODE*);  
   }  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
    
   return complaintID;  
   }  
    
   public void rewriteComplaintsANDRecomendationInBD(String complaint, String recomendation, String numberOfCard, String doctorFIO, String date, String time, int tableNumber)  
   {  
   String update = "UPDATE " + CONST.*COMPLAINTS\_TABLE* + " SET " + CONST.*COMPLAINTS\_COMPLAINTS* + " = '" + complaint + "'" +  
   " WHERE " + CONST.*COMPLAINTS\_NUMBER\_OF\_CARD* + " = " + numberOfCard +  
   " AND " + CONST.*COMPLAINTS\_DATE\_OF\_RECEPTION* + " = DATE '" + date + "' AND " + CONST.*COMPLAINTS\_TIME\_OF\_RECEPTION* + " = '" + time + "'";  
    
   String update2 = "UPDATE " + CONST.*RECOMENDATION\_TABLE* + " SET " + CONST.*RECOMENDATION\_RECOMENDATION* + " = '" + recomendation + "'" +  
   " WHERE " + CONST.*RECOMENDATION\_RECOMENDATION\_CODE* + " = " + getComplaintsID(date, time, numberOfCard);  
    
   Statement st = null;  
   Statement st2 = null;  
   try  
   {  
   st = getDbConnecion().createStatement();  
   st2 = getDbConnecion().createStatement();  
    
   st.executeUpdate(update);  
   st2.executeUpdate(update2);  
   } catch (SQLException e) { e.printStackTrace(); } catch (ClassNotFoundException e) { e.printStackTrace(); }  
   finally  
   {  
   try {  
   dbDisconnectionSt(st);  
   dbDisconnectionSt(st2);  
   } catch (SQLException e) {  
   e.printStackTrace();  
   }  
   }  
   }  
    
   public ResultSet getAllReceptionDoneToDoctor(Statement statement, int tableNumber)  
   {  
   String select =  
   "SELECT rec." + CONST.*RECEPTION\_DATE\_OF\_RECEPTION* + ", rec." + CONST.*RECEPTION\_TIME\_OF\_RECEPTION* + ", rec." + CONST.*RECEPTION\_NUMBER\_OF\_CARD* + ", CONCAT(p." +  
   CONST.*PATIENT\_SECONDNAME* + ", ' ', p." + CONST.*PATIENT\_FIRSTNAME* + ", ' ', p." + CONST.*PATIENT\_THIRDNAME* + "), pl." + CONST.*PLOTS\_NUMBER* + ", rec." + CONST.*RECEPTION\_IS\_VISIT* +  
   " FROM " + CONST.*RECEPTION\_TABLE* + " rec INNER JOIN " + CONST.*DOCTOR\_TABLE* + " doc INNER JOIN " + CONST.*SPECIALIZATION\_TABLE* + " sp INNER JOIN " + CONST.*PATIENT\_TABLE* + " p INNER JOIN " + CONST.*PLOT\_TABLE* + " pl" +  
   " ON doc." + CONST.*DOCTORS\_TABLE\_NUMBER* + " = rec." + CONST.*RECEPTION\_TABLE\_NUMBER* + " AND sp." + CONST.*SPECIALIZATION\_CODE* + " = doc." + CONST.*DOCTORS\_SPECIALIZATION\_CODE* + " AND doc." +  
   CONST.*DOCTORS\_TABLE\_NUMBER* + " = pl." + CONST.*PLOTS\_DOCTORS\_TABLE\_NUMBER* + " AND p." + CONST.*PATIENT\_NUMBER\_OF\_CARD* + " = rec." + CONST.*RECEPTION\_NUMBER\_OF\_CARD* +  
   " WHERE rec." + CONST.*RECEPTION\_TABLE\_NUMBER* + " = " + String.*valueOf*(tableNumber) + " AND rec." + CONST.*RECEPTION\_IS\_VISIT* + " = true";  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(select);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public ResultSet getAllReceptionToDoctor(Statement statement, int tableNumber)  
   {  
   String select =  
   "SELECT rec." + CONST.*RECEPTION\_DATE\_OF\_RECEPTION* + ", rec." + CONST.*RECEPTION\_TIME\_OF\_RECEPTION* + ", rec." + CONST.*RECEPTION\_NUMBER\_OF\_CARD* + ", CONCAT(p." +  
   CONST.*PATIENT\_SECONDNAME* + ", ' ', p." + CONST.*PATIENT\_FIRSTNAME* + ", ' ', p." + CONST.*PATIENT\_THIRDNAME* + "), pl." + CONST.*PLOTS\_NUMBER* + ", rec." + CONST.*RECEPTION\_IS\_VISIT* +  
   " FROM " + CONST.*RECEPTION\_TABLE* + " rec INNER JOIN " + CONST.*DOCTOR\_TABLE* + " doc INNER JOIN " + CONST.*SPECIALIZATION\_TABLE* + " sp INNER JOIN " + CONST.*PATIENT\_TABLE* + " p INNER JOIN " + CONST.*PLOT\_TABLE* + " pl" +  
   " ON doc." + CONST.*DOCTORS\_TABLE\_NUMBER* + " = rec." + CONST.*RECEPTION\_TABLE\_NUMBER* + " AND sp." + CONST.*SPECIALIZATION\_CODE* + " = doc." + CONST.*DOCTORS\_SPECIALIZATION\_CODE* + " AND doc." +  
   CONST.*DOCTORS\_TABLE\_NUMBER* + " = pl." + CONST.*PLOTS\_DOCTORS\_TABLE\_NUMBER* + " AND p." + CONST.*PATIENT\_NUMBER\_OF\_CARD* + " = rec." + CONST.*RECEPTION\_NUMBER\_OF\_CARD* +  
   " WHERE rec." + CONST.*RECEPTION\_TABLE\_NUMBER* + " = " + String.*valueOf*(tableNumber) + " AND rec." + CONST.*RECEPTION\_IS\_VISIT* + " = false";  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(select);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public ResultSet getAllReception(Statement statement)  
   {  
   String select =  
   "SELECT rec." + CONST.*RECEPTION\_TABLE\_NUMBER* + ", sp." + CONST.*SPECIALIZATION* + ", CONCAT(doc." + CONST.*DOCTORS\_SECONDNAME* + ", ' ', doc." + CONST.*DOCTORS\_FIRSTNAME* + ", ' ', doc." + CONST.*DOCTORS\_THIRDNAME* + "), rec." +  
   CONST.*RECEPTION\_DATE\_OF\_RECEPTION* + ", rec." + CONST.*RECEPTION\_TIME\_OF\_RECEPTION* + ", rec." + CONST.*RECEPTION\_NUMBER\_OF\_CARD* + ", CONCAT(p." +  
   CONST.*PATIENT\_SECONDNAME* + ", ' ', p." + CONST.*PATIENT\_FIRSTNAME* + ", ' ', p." + CONST.*PATIENT\_THIRDNAME* + "), pl." + CONST.*PLOTS\_NUMBER* + ", rec." + CONST.*RECEPTION\_IS\_VISIT* +  
   " FROM " + CONST.*RECEPTION\_TABLE* + " rec INNER JOIN " + CONST.*DOCTOR\_TABLE* + " doc INNER JOIN " + CONST.*SPECIALIZATION\_TABLE* + " sp INNER JOIN " + CONST.*PATIENT\_TABLE* + " p INNER JOIN " + CONST.*PLOT\_TABLE* + " pl" +  
   " WHERE doc." + CONST.*DOCTORS\_TABLE\_NUMBER* + " = rec." + CONST.*RECEPTION\_TABLE\_NUMBER* + " AND sp." + CONST.*SPECIALIZATION\_CODE* + " = doc." + CONST.*DOCTORS\_SPECIALIZATION\_CODE* + " AND doc." +  
   CONST.*DOCTORS\_TABLE\_NUMBER* + " = pl." + CONST.*PLOTS\_DOCTORS\_TABLE\_NUMBER* + " AND p." + CONST.*PATIENT\_NUMBER\_OF\_CARD* + " = rec." + CONST.*RECEPTION\_NUMBER\_OF\_CARD* + " AND rec." + CONST.*RECEPTION\_IS\_VISIT* + " = false";  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(select);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public ResultSet getReceptionForCheck(int tableNumber, String date, Statement statement)  
   {  
   String select = "SELECT \* FROM " + CONST.*RECEPTION\_TABLE* +  
   " WHERE " + CONST.*RECEPTION\_TABLE\_NUMBER* + " = " + tableNumber +  
   " AND " + CONST.*RECEPTION\_DATE\_OF\_RECEPTION* + " = DATE '" + date + "'";  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(select);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public ArrayList getAllTime()  
   {  
   ArrayList timeList = new ArrayList();  
    
   String select = "SELECT \* FROM " + CONST.*TIME\_TABLE*;  
    
   Statement st = null;  
   ResultSet resSet = null;  
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select); // Выполнить SQL запрос о взятии данных из БД (и записать результат в переменную)  
   } catch (SQLException e) {  
   e.printStackTrace();  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
    
   try  
   {  
   while(resSet.next())  
   {  
   ArrayList temp = new ArrayList();  
   temp.add(resSet.getInt(CONST.*TIME\_ID*));  
   temp.add(resSet.getString(CONST.*TIME\_HOURS*));  
    
   timeList.add(temp);  
   }  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
    
   return timeList;  
   }  
    
   public Vector<String> getTime()  
   {  
   Vector<String> timeList = new Vector();  
    
   String select = "SELECT " + CONST.*TIME\_HOURS* + " FROM " + CONST.*TIME\_TABLE*;  
    
   Statement st = null;  
   ResultSet resSet = null;  
   try  
   {  
   st = getDbConnecion().createStatement();  
   resSet = st.executeQuery(select); // Выполнить SQL запрос о взятии данных из БД (и записать результат в переменную)  
   } catch (SQLException e) {  
   e.printStackTrace();  
   } catch (ClassNotFoundException e) {  
   e.printStackTrace();  
   }  
    
   try  
   {  
   while(resSet.next())  
   {  
   timeList.add(resSet.getString(CONST.*TIME\_HOURS*));  
   }  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionSt(st);  
   dbDisconnectionResSet(resSet);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
    
   return timeList;  
   }  
    
   public ResultSet getDoctorsForReferenceTable(Statement statement)  
   {  
   String select = "SELECT " + "doc." + CONST.*DOCTORS\_TABLE\_NUMBER* + "," + CONST.*DOCTORS\_SECONDNAME* + "," + CONST.*DOCTORS\_FIRSTNAME* + "," + CONST.*DOCTORS\_THIRDNAME* + "," +  
   CONST.*DOCTORS\_GENDER* + "," + CONST.*DOCTORS\_BIRTHDAY* + "," +  
   "sp." + CONST.*SPECIALIZATION* + "," + "sp." + CONST.*SPECIALIZATION\_CODE* + "," +  
   "doc." + CONST.*DOCTORS\_DATE\_START\_JOB* + "," + "pl." + CONST.*PLOTS\_NUMBER* +  
   " FROM " + CONST.*DOCTOR\_TABLE* + " doc INNER JOIN " + CONST.*SPECIALIZATION\_TABLE* + " sp INNER JOIN " + CONST.*PLOT\_TABLE* + " pl" +  
   " ON doc." + CONST.*DOCTORS\_TABLE\_NUMBER* + "=pl." + CONST.*PLOTS\_DOCTORS\_TABLE\_NUMBER* +  
   " WHERE doc." + CONST.*DOCTORS\_SPECIALIZATION\_CODE* + "=sp." + CONST.*SPECIALIZATION\_CODE*;  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(select);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public ResultSet getPatientsForReferenceTable(Statement statement)  
   {  
   String select = "SELECT " + CONST.*PATIENT\_SECONDNAME* + ", " + CONST.*PATIENT\_FIRSTNAME* + ", " + CONST.*PATIENT\_THIRDNAME* + ", " +  
   CONST.*PATIENT\_GENDER* + ", " + CONST.*PATIENT\_BIRTHDAY* + ", " + CONST.*PATIENT\_NUMBER\_OF\_PLOT* + ", " +  
   CONST.*PATIENT\_DATE\_OF\_CREATE\_CARD* + ", " + CONST.*PATIENT\_NUMBER\_OF\_CARD* + ", " + CONST.*PATIENT\_NUMBER\_OF\_PASPORT* + ", " +  
   CONST.*PATIENT\_NUMBER\_OF\_PHONE* + ", " + CONST.*PATIENT\_INSURANCE\_POLICY* + ", " +  
   CONST.*PATIENT\_STREET* + ", " + CONST.*PATIENT\_BUILDING* + ", " + CONST.*PATIENT\_APARTMENT* +  
   " FROM " + CONST.*PATIENT\_TABLE*;  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(select);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public ResultSet getTimeBeginEnd(Statement statement, int tableNumber)  
   {  
   String join = "SELECT " + CONST.*TIMEOFJOB\_ID\_TIME\_BEGIN* + ", " + CONST.*TIMEOFJOB\_ID\_TIME\_END* + " FROM " + CONST.*TIMEOFJOB\_TABLE*;  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(join);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public ResultSet getTimetableForReferenceTable(Statement statement)  
   {  
   String join = "SELECT CONCAT(doc." + CONST.*DOCTORS\_SECONDNAME* + ", ' ' , doc." + CONST.*DOCTORS\_FIRSTNAME* + ", ' ', doc." + CONST.*DOCTORS\_THIRDNAME* + "), " +  
   "sp." + CONST.*SPECIALIZATION* + ", tt." + CONST.*CABINETS\_NUMBER* + ", d." + CONST.*DAY\_DAY* + ", t." + CONST.*TIME\_HOURS* + ", ti." + CONST.*TIME\_HOURS* + ", w." + CONST.*WEEK\_DATE* + ", doc." + CONST.*DOCTORS\_TABLE\_NUMBER* + ", tj." + CONST.*TIMEOFJOB\_ID\_WEEK* +  
   " FROM " + CONST.*DOCTOR\_TABLE* + " doc INNER JOIN " + CONST.*TIMETABLE\_TABLE* + " tt INNER JOIN " + CONST.*SPECIALIZATION\_TABLE* + " sp" +  
   " INNER JOIN " + CONST.*DAY\_TABLE* + " d INNER JOIN " + CONST.*TIMETABLEDAY\_TABLE* + " ttd INNER JOIN " + CONST.*TIMETABLETIME\_TABLE* + " ttt" +  
   " INNER JOIN " + CONST.*TIME\_TABLE* + " t INNER JOIN " + CONST.*TIME\_TABLE* + " ti INNER JOIN " + CONST.*CABINETS\_TABLE* + " cab INNER JOIN " + CONST.*WEEK\_TABLE* + " w INNER JOIN " + CONST.*TIMEOFJOB\_TABLE* + " tj" +  
   " ON tt." + CONST.*TIMETABLE\_TABLE\_NUMBER* + " = doc." + CONST.*DOCTORS\_TABLE\_NUMBER* +  
   " AND sp." + CONST.*SPECIALIZATION\_CODE* + " = doc." + CONST.*DOCTORS\_SPECIALIZATION\_CODE* +  
   " AND cab." + CONST.*CABINETS\_NUMBER* + " = tt." + CONST.*TIMETABLE\_NUMBER\_CABINET* +  
   " AND ttd." + CONST.*TIMETABLEDAY\_TABLE\_NUMBER* + " = tt." + CONST.*TIMETABLE\_TABLE\_NUMBER* +  
   " AND ttd." + CONST.*TIMETABLEDAY\_ID\_DAY* + " = d." + CONST.*DAY\_ID* +  
   " AND ttt." + CONST.*TIMETABLETIME\_ID\_DAY* + " = ttd." + CONST.*TIMETABLEDAY\_ID\_DAY* +  
   " AND ttt." + CONST.*TIMETABLETIME\_TABLE\_NUMBER* + " = tt." + CONST.*TIMETABLE\_TABLE\_NUMBER* +  
   " AND tj." + CONST.*TIMEOFJOB\_ID\_TIME\_BEGIN* + " = t." + CONST.*TIME\_ID* +  
   " AND tj." + CONST.*TIMEOFJOB\_ID\_TIME\_END* + " = ti." + CONST.*TIME\_ID* +  
   " AND w." + CONST.*WEEK\_ID\_DAY* + " = d." + CONST.*DAY\_ID* +  
   " AND tj." + CONST.*TIMEOFJOB\_TABLE\_NUMBER* + " = ttt." + CONST.*TIMETABLETIME\_TABLE\_NUMBER* +  
   " AND tj." + CONST.*TIMEOFJOB\_ID\_WEEK* + " = w." + CONST.*WEEK\_ID* +  
   " ORDER BY doc." + CONST.*DOCTORS\_TABLE\_NUMBER* + ", w." + CONST.*WEEK\_ID\_DAY*;  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(join);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public ResultSet getWeek(Statement statement)  
   {  
   String select = "SELECT \* FROM " + CONST.*WEEK\_TABLE*;  
    
   ResultSet resSet = null;  
   Statement st = statement;  
    
   try  
   {  
   resSet = st.executeQuery(select);  
   }  
   catch (SQLException e) // Если произошла ошибка в самой БД (Не создан Connection)  
   {  
   System.*out*.println(e);  
   }  
    
   return resSet;  
   }  
    
   public void rewriteDaysInWeeks()  
   {  
   Statement st = null;  
   CallableStatement cs = null;  
   //String execute = "DROP TABLE IF EXISTS temp; CREATE TABLE temp SELECT \* FROM " + CONST.TIMEOFJOB\_TABLE + "; SET SQL\_SAFE\_UPDATES = 0; CALL " + CONST.PROCEDURE\_OFFSET\_FINAL + "; DEALLOCATE cur; SET SQL\_SAFE\_UPDATES = 1;";  
    
   try  
   {  
   cs = getDbConnecion().prepareCall("{ CALL " + CONST.*PROCEDURE\_OFFSET\_FINAL* + "}");  
   cs.execute();  
   //st = getDbConnecion().createStatement();  
   //st.executeUpdate(execute);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   catch (ClassNotFoundException e)  
   {  
   e.printStackTrace();  
   }  
   finally // Нужно обязательно закрыть подключение  
   {  
   try  
   {  
   dbDisconnectionCs(cs);  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   }  
   }  
   }  
    
   public boolean executeDB(ArrayList<String> executeList)  
   {  
   Statement st = null;  
    
   try  
   {  
   st = getDbConnecion().createStatement();  
   for(String execute : executeList) // Поочерёдно выполнять обновление строк БД  
   {  
   st.executeUpdate(execute);  
   }  
   }  
   catch (SQLException e)  
   {  
   e.printStackTrace();  
   try  
   {  
   dbDisconnectionSt(st);  
   } catch (SQLException ee) { ee.printStackTrace(); return false; }  
   return false;  
   }  
   catch (ClassNotFoundException e)  
   {  
   e.printStackTrace();  
   try  
   {  
   dbDisconnectionSt(st);  
   } catch (SQLException ee) { ee.printStackTrace(); return false; }  
   return false;  
   }  
    
   return true;  
   }  
  }