package com.example.system\_testing;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import java.io.IOException;  
  
*/\*\*  
 \* Главный класс, с которого начинается программа.  
 \*/*public class Authentication extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Authentication.class.getResource("authentication.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 700, 400);  
 stage.setTitle("System testing");  
 stage.setScene(scene);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

package com.example.system\_testing.auxiliary;  
  
*/\*\*  
 \* Класс содержит константы имён всех пользовательских окон формата fxml.  
 \*/*public class ConstNameWindows {  
  
 public static final String *WINDOW\_AUTHENTICATION* = "authentication.fxml";  
 public static final String *WINDOW\_ADMINISTRATOR\_MENU* = "administratorMenu.fxml";  
 public static final String *WINDOW\_CHANGE\_DISCIPLINES* = "changeDisciplines.fxml";  
 public static final String *WINDOW\_CHANGE\_GROUPS* = "changeGroups.fxml";  
 public static final String *WINDOW\_DELETE\_USER* = "deleteUser.fxml";  
 public static final String *WINDOW\_REG\_STUDENT* = "regStudent.fxml";  
 public static final String *WINDOW\_REG\_TEACHER* = "regTeacher.fxml";  
 public static final String *WINDOW\_SHOW\_RESULT\_TEST* = "showResultTest.fxml";  
 public static final String *WINDOW\_STUDENT\_MENU* = "studentMenu.fxml";  
 public static final String *WINDOW\_TEACHER\_MENU* = "teacherMenu.fxml";  
 public static final String *WINDOW\_APPOINT\_DATE\_TEST* = "appointDateTest.fxml";  
 public static final String *WINDOW\_CHANGE\_DROP\_TEST* = "changeDropTest.fxml";  
 public static final String *WINDOW\_CREATE\_NEW\_TEST* = "createNewTest.fxml";  
 public static final String *WINDOW\_CREATE\_QUESTION* = "createQuestion.fxml";  
 public static final String *WINDOW\_PASS\_TEST* = "passTest.fxml";  
  
}

package com.example.system\_testing.auxiliary;  
  
*/\*\*  
 \* Класс содержит константы всех имён таблиц и столбцов.  
 \*/*public class ConstTables {  
  
 public static final String *URL\_PACKAGE* = "/com/example/system\_testing/";  
  
 public static final String *USERS\_TABLE* = "tbl\_users";  
 public static final String *USERS\_ID* = "id";  
 public static final String *USERS\_LOGIN* = "login";  
 public static final String *USERS\_PASSWORD* = "password";  
 public static final String *USERS\_ROLE* = "role";  
  
 public static final String *TEACHERS\_TABLE* = "tbl\_teachers";  
 public static final String *TEACHERS\_ID* = "id";  
 public static final String *TEACHERS\_FIO* = "fio";  
 public static final String *TEACHERS\_USER\_ID* = "tbl\_users\_id";  
  
 public static final String *STUDENTS\_TABLE* = "tbl\_students";  
 public static final String *STUDENTS\_ID* = "id";  
 public static final String *STUDENTS\_FIO* = "fio";  
 public static final String *STUDENTS\_USER\_ID* = "tbl\_users\_id";  
 public static final String *STUDENTS\_GROUPS\_ID* = "tbl\_groups\_id";  
  
 public static final String *ANSWERS\_TABLE* = "tbl\_answers";  
 public static final String *ANSWERS\_ID* = "id";  
 public static final String *ANSWERS\_NAME* = "name";  
 public static final String *ANSWERS\_IS\_TRUE* = "is\_true";  
 public static final String *ANSWERS\_QUESTION\_ID* = "tbl\_questions\_id";  
  
 public static final String *QUESTIONS\_TABLE* = "tbl\_questions";  
 public static final String *QUESTIONS\_ID* = "id";  
 public static final String *QUESTIONS\_NAME* = "name";  
 public static final String *QUESTIONS\_TESTES\_ID* = "tbl\_testes\_id";  
  
 public static final String *DISCIPLINES\_TABLE* = "tbs\_disciplines";  
 public static final String *DISCIPLINES\_ID* = "id";  
 public static final String *DISCIPLINES\_NAME* = "name";  
  
 public static final String *TESTS\_TABLE* = "tbl\_testes";  
 public static final String *TESTS\_ID* = "id";  
 public static final String *TESTS\_NAME* = "name";  
 public static final String *TESTS\_DISCIPLINES\_ID* = "tbs\_disciplines\_id";  
  
 public static final String *GROUPS\_TABLE* = "tbl\_groups";  
 public static final String *GROUPS\_ID* = "id";  
 public static final String *GROUPS\_NUMBER* = "number";  
  
 public static final String *ASSESSMENTS\_TABLE* = "tbl\_assessments";  
 public static final String *ASSESSMENTS\_ID* = "id";  
 public static final String *ASSESSMENTS\_ASSESSMENT* = "assessment";  
 public static final String *ASSESSMENTS\_STUDENT\_ID* = "tbl\_students\_id";  
 public static final String *ASSESSMENTS\_TESTES\_ID* = "tbl\_testes\_id";  
  
 public static final String *SCHEDULES\_TABLE* = "tbl\_schedules";  
 public static final String *SCHEDULES\_ID* = "id";  
 public static final String *SCHEDULES\_DATE* = "date";  
 public static final String *SCHEDULES\_GROUPS\_ID* = "tbl\_groups\_id";  
 public static final String *SCHEDULES\_TESTES\_ID* = "tbl\_testes\_id";  
  
 public static final String *DISCIPLINES\_HAS\_TEACHERS\_TABLE* = "tbl\_teachers\_has\_tbs\_disciplines";  
 public static final String *DISCIPLINES\_HAS\_TEACHERS\_DISCIPLINES\_ID* = "tbs\_disciplines\_id";  
 public static final String *DISCIPLINES\_HAS\_TEACHERS\_TEACHERS\_ID* = "tbl\_teachers\_id";  
  
}

package com.example.system\_testing.auxiliary;  
  
import javafx.animation.TranslateTransition;  
import javafx.scene.Node;  
import javafx.util.Duration;  
  
*/\*\*  
 \* Класс позволяет создать объект с вызовом функции анимации.  
 \*/*public class Shake {  
 private TranslateTransition tt;  
  
 public Shake(Node node) {  
 tt = new TranslateTransition(Duration.*millis*(70), node);  
 tt.setFromX(0f);  
 tt.setByX(10f);  
 tt.setCycleCount(3);  
 tt.setAutoReverse(true);  
 }  
  
 public void playAnim() {  
 tt.playFromStart();  
 }  
}

package com.example.system\_testing.auxiliary;  
  
import com.example.system\_testing.controller.\*;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.Test;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.util.ArrayList;  
  
*/\*\*  
 \* В классе содержатся методы для перехода между пользовательскими формами и передачи данных между ними.  
 \*/*public class WorkWithScene {  
  
 public void getNewWindow (String window) {  
  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource(ConstTables.*URL\_PACKAGE* + window));  
  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setTitle("System testing");  
 stage.setScene(new Scene(root));  
 stage.show();  
  
 }  
  
 public void getNewWindow (String window, Test test) {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource(ConstTables.*URL\_PACKAGE* + window));  
  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setTitle("System testing");  
 stage.setScene(new Scene(root));  
  
 CreateQuestionController createQuestionController = loader.getController();  
 createQuestionController.setNameTest(test);  
  
 stage.show();  
 }  
  
 public void getNewWindow (String window, int id) {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 FXMLLoader loader = new FXMLLoader();  
  
 loader.setLocation(getClass().getResource(ConstTables.*URL\_PACKAGE* + window));  
  
 try {  
 loader.load();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setTitle("System testing");  
 stage.setScene(new Scene(root));  
  
 String userRole = dbHandler.getUserRole(id);  
 if (userRole.equals("teacher")) {  
 TeacherMenuController teacherMenuController = loader.getController();  
 teacherMenuController.setUserID(id);  
 } else if (userRole.equals("student")) {  
 StudentMenuController studentMenuController = loader.getController();  
 studentMenuController.setUserID(id);  
 }  
  
 stage.show();  
  
 }  
  
 public void getNewWindow (String window, ArrayList<String> list) {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource(ConstTables.*URL\_PACKAGE* + window));  
  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setTitle("System testing");  
 stage.setScene(new Scene(root));  
  
 CreateNewTestController createNewTestController = loader.getController();  
 createNewTestController.setDisciplinesList(list);  
  
 stage.show();  
 }  
  
 public void getNewWindow (String window, String str, int userID) {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource(ConstTables.*URL\_PACKAGE* + window));  
  
 try {  
 loader.load();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 Parent root = loader.getRoot();  
 Stage stage = new Stage();  
 stage.setTitle("System testing");  
 stage.setScene(new Scene(root));  
  
 PassTestController passTestController = loader.getController();  
 passTestController.setNameTestAndUserID(str, userID);  
  
 stage.show();  
 }  
  
}

package com.example.system\_testing.controller;  
  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой меню администратора.  
 \*/*public class AdministratorMenuController {  
  
 WorkWithScene ws = new WorkWithScene();  
  
 @FXML  
 private Button changeGroups\_button;  
  
 @FXML  
 private Button changeDisciplines\_button;  
  
 @FXML  
 private Button deleteUser\_button;  
  
 @FXML  
 private Button exitFromSystem\_button;  
  
 @FXML  
 private Button regStudent\_button;  
  
 @FXML  
 private Button regTeacher\_button;  
  
 @FXML  
 private Button showResultTest\_button;  
  
 @FXML  
 void initialize() {  
  
 regTeacher\_button.setOnAction(event -> {  
 regTeacher();  
 });  
  
 regStudent\_button.setOnAction(event -> {  
 regStudent();  
 });  
  
 deleteUser\_button.setOnAction(event -> {  
 deleteUser();  
 });  
  
 showResultTest\_button.setOnAction(event -> {  
 showResultTest();  
 });  
  
 changeDisciplines\_button.setOnAction(event -> {  
 changeDisciplines();  
 });  
  
 changeGroups\_button.setOnAction(event -> {  
 changeGroups();  
 });  
  
 exitFromSystem\_button.setOnAction(event -> {  
 exitFromSystem();  
 });  
  
 }  
  
 private void regTeacher() {  
 regTeacher\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_REG\_TEACHER*);  
 }  
  
 private void regStudent() {  
 regStudent\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_REG\_STUDENT*);  
 }  
  
 private void deleteUser() {  
 deleteUser\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_DELETE\_USER*);  
 }  
  
 private void showResultTest(){  
 showResultTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_SHOW\_RESULT\_TEST*);  
 }  
  
 private void changeDisciplines() {  
 changeDisciplines\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CHANGE\_DISCIPLINES*);  
 }  
  
 private void changeGroups() {  
 changeGroups\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CHANGE\_GROUPS*);  
 }  
  
 private void exitFromSystem() {  
 exitFromSystem\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_AUTHENTICATION*);  
 }  
  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.DatePicker;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой назначения даты теста.  
 \*/*public class AppointDateTestController {  
  
 WorkWithScene ws = new WorkWithScene();  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 String nameTest;  
 String numberGroup;  
 String date;  
  
 @FXML  
 private Button appointTest\_button;  
  
 @FXML  
 private ComboBox<String> choiceGroup\_comboBox;  
  
 @FXML  
 private ComboBox<String> choiceTest\_comboBox;  
  
 @FXML  
 private DatePicker dateTest\_datePicker;  
  
 @FXML  
 private Button goBack\_button;  
  
 @FXML  
 void selectDate(ActionEvent event) {  
 date = dateTest\_datePicker.getValue().toString();  
 }  
  
 @FXML  
 void selectTest(ActionEvent event) {  
 nameTest = choiceTest\_comboBox.getSelectionModel().getSelectedItem();  
 }  
  
 @FXML  
 void selectGroup(ActionEvent event) {  
 numberGroup = choiceGroup\_comboBox.getSelectionModel().getSelectedItem();  
 }  
  
 @FXML  
 void initialize() {  
  
 ObservableList<String> testsList = FXCollections.*observableArrayList*(choiceTest());  
 choiceTest\_comboBox.setItems(testsList);  
  
 ObservableList<String> groupsList = FXCollections.*observableArrayList*(choiceGroups());  
 choiceGroup\_comboBox.setItems(groupsList);  
  
  
 appointTest\_button.setOnAction(event -> {  
 appointTest();  
 });  
  
 goBack\_button.setOnAction(event -> {  
 goBack();  
 });  
  
 }  
  
 private ArrayList<String> choiceTest() {  
 return dbHandler.getTestList();  
 }  
  
 private ArrayList<String> choiceGroups() {  
 return dbHandler.getGroupsList();  
 }  
  
 private void appointTest() {  
  
 dbHandler.appointDateInDB(date, nameTest, numberGroup);  
  
 appointTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_APPOINT\_DATE\_TEST*);  
  
 }  
  
 private void goBack() {  
 goBack\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_TEACHER\_MENU*);  
 }  
  
}

package com.example.system\_testing.controller;  
  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import com.example.system\_testing.auxiliary.ConstTables;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.Shake;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.User;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.PasswordField;  
import javafx.scene.control.TextField;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой идентификации пользователя.  
 \*/*public class AuthenticationController {  
  
 WorkWithScene ws = new WorkWithScene();  
  
 @FXML  
 private Button authSigIn\_button;  
  
 @FXML  
 private TextField login\_field;  
  
 @FXML  
 private PasswordField password\_field;  
  
 @FXML  
 void initialize() {  
  
 authSigIn\_button.setOnAction(event -> {  
 String login = login\_field.getText();  
 String password = password\_field.getText();  
  
 if (!login.equals("") && !password.equals("")) {  
 loginUser(login, password);  
 } else {  
 Shake userLoginAnim = new Shake(login\_field);  
 Shake userPassAnim = new Shake(password\_field);  
 userLoginAnim.playAnim();  
 userPassAnim.playAnim();  
 }  
 });  
  
 }  
  
 private void loginUser(String login, String password) {  
  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 ResultSet resultSet = null;  
 int counter = 0;  
  
 User user = new User();  
 user.setUserLogin(login);  
 user.setUserPassword(password);  
  
 resultSet = dbHandler.getUser(user);  
  
 while (true) {  
 try {  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 if (resultSet.getString(ConstTables.*USERS\_ROLE*).equalsIgnoreCase("administrator")) {  
 authSigIn\_button.getScene().getWindow().hide();  
 counter++;  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 } else if (resultSet.getString(ConstTables.*USERS\_ROLE*).equalsIgnoreCase("teacher")) {  
 authSigIn\_button.getScene().getWindow().hide();  
 counter++;  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_TEACHER\_MENU*, resultSet.getInt(ConstTables.*USERS\_ID*));  
 } else if (resultSet.getString(ConstTables.*USERS\_ROLE*).equalsIgnoreCase("student")) {  
 authSigIn\_button.getScene().getWindow().hide();  
 counter++;  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_STUDENT\_MENU*, resultSet.getInt(ConstTables.*USERS\_ID*));  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if (counter == 0) {  
 Shake userLoginAnim = new Shake(login\_field);  
 Shake userPassAnim = new Shake(password\_field);  
 userLoginAnim.playAnim();  
 userPassAnim.playAnim();  
 }  
 }  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.TextField;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой изменения списка дисциплин: добавление новой / удаление существующей.  
 \*/*public class ChangeDisciplinesController {  
  
 WorkWithScene ws = new WorkWithScene();  
  
 @FXML  
 private Button addDiscipline\_button;  
  
 @FXML  
 private ComboBox<String> choiceDiscipline\_comboBox;  
  
 @FXML  
 private Button deleteDiscipline\_button;  
  
 @FXML  
 private Button goBack\_button;  
  
 @FXML  
 private TextField nameDiscipline\_textField;  
  
 @FXML  
 void initialize() {  
  
 ObservableList<String> disciplinesList = FXCollections.*observableArrayList*(choiceDisciplines());  
 choiceDiscipline\_comboBox.setItems(disciplinesList);  
  
 choiceDiscipline\_comboBox.setOnAction(event -> {  
  
 if(!(choiceDiscipline\_comboBox.getSelectionModel().getSelectedItem().equals(""))) {  
 deleteDiscipline\_button.setDisable(false);  
 }  
 });  
  
 nameDiscipline\_textField.setOnAction(event -> {  
 String str = nameDiscipline\_textField.getText();  
 if (!(str.equals(""))) {  
 addDiscipline\_button.setDisable(false);  
 }  
 });  
  
 addDiscipline\_button.setOnAction(event -> {  
  
 addDisciplineToBD();  
 addDiscipline\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 });  
  
 goBack\_button.setOnAction(event -> {  
 goBack();  
 });  
  
 deleteDiscipline\_button.setOnAction(event -> {  
 delete();  
 deleteDiscipline\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 });  
  
 }  
  
 private void delete() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 String discipline = choiceDiscipline\_comboBox.getSelectionModel().getSelectedItem();  
 dbHandler.deleteDiscipline(discipline);  
 }  
  
 private void addDisciplineToBD() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 String discipline = nameDiscipline\_textField.getText();  
 dbHandler.addDiscipline(discipline);  
 }  
  
 private void goBack() {  
 goBack\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 }  
  
 private ArrayList<String> choiceDisciplines() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getDisciplinesList();  
 }  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.Question;  
import com.example.system\_testing.essences.Test;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.text.Text;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой изменения списков вопросов и удаления тестов.  
 \*/*public class ChangeDropTestController {  
  
 WorkWithScene ws = new WorkWithScene();  
 String nameTest;  
 ArrayList<String> questionsList = new ArrayList<>();  
 Test test;  
  
 @FXML  
 private Button addQuestion\_button;  
  
 @FXML  
 private ComboBox<String> choiceTest\_comboBox;  
  
 @FXML  
 private ComboBox<String> choiceQuestion\_comboBox;  
  
 @FXML  
 private Button deleteQuestion\_button;  
  
 @FXML  
 private Button deleteTest\_button;  
  
 @FXML  
 private Button goBack\_button;  
  
 @FXML  
 private Text nameAnswerFour\_text;  
  
 @FXML  
 private Text nameAnswerOne\_text;  
  
 @FXML  
 private Text nameAnswerThree\_text;  
  
 @FXML  
 private Text nameAnswerTwo\_text;  
  
 @FXML  
 private Text nameQuestion\_text;  
  
 @FXML  
 void initialize() {  
  
 ObservableList<String> testList = FXCollections.*observableArrayList*(choiceTest());  
 choiceTest\_comboBox.setItems(testList);  
  
 choiceTest\_comboBox.setOnAction(event -> {  
 nameTest = choiceTest\_comboBox.getSelectionModel().getSelectedItem();  
 questionsList = choiceQuestion();  
  
 ObservableList<String> questionList = FXCollections.*observableArrayList*(questionsList);  
 choiceQuestion\_comboBox.setItems(questionList);  
  
 choiceQuestion\_comboBox.setOnAction(events -> {  
  
 String nameQuestion = choiceQuestion\_comboBox.getSelectionModel().getSelectedItem();  
 nameQuestion\_text.setText(nameQuestion);  
  
 if (!(nameQuestion.equals(""))) {  
 Question question = new Question();  
 question = getQuestion(nameQuestion);  
  
 nameAnswerOne\_text.setText(question.getAnswerList().get(0).getNameAnswer() + "\t" +  
 question.getAnswerList().get(0).getIsTrueAnswer());  
 nameAnswerTwo\_text.setText(question.getAnswerList().get(1).getNameAnswer() + "\t" +  
 question.getAnswerList().get(1).getIsTrueAnswer());  
 nameAnswerThree\_text.setText(question.getAnswerList().get(2).getNameAnswer() + "\t" +  
 question.getAnswerList().get(2).getIsTrueAnswer());  
 nameAnswerFour\_text.setText(question.getAnswerList().get(3).getNameAnswer() + "\t" +  
 question.getAnswerList().get(3).getIsTrueAnswer());  
 }  
  
 });  
  
 });  
  
 addQuestion\_button.setOnAction(event -> {  
 goToWindowsAddQuestion();  
 });  
  
 deleteQuestion\_button.setOnAction(event -> {  
 deleteQuestion();  
 });  
  
 deleteTest\_button.setOnAction(event -> {  
 deleteTest();  
 });  
  
 goBack\_button.setOnAction(event -> {  
 goBack();  
 });  
  
 }  
  
 private Question getQuestion(String nameQuestion) {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getQuestion(nameQuestion);  
 }  
  
 private ArrayList<String> choiceQuestion() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getQuestionsList(nameTest);  
 }  
  
 private ArrayList<String> choiceTest() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getTestList();  
 }  
  
 private void goToWindowsAddQuestion() {  
  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 test = dbHandler.getTest(nameTest);  
 addQuestion\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CREATE\_QUESTION*, test);  
 }  
  
 private void deleteQuestion() {  
  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 String nameQuestion = choiceQuestion\_comboBox.getSelectionModel().getSelectedItem();  
 dbHandler.deleteQuestion(nameQuestion);  
  
 deleteTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CHANGE\_DROP\_TEST*);  
 }  
  
 private void deleteTest() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 nameTest = choiceTest\_comboBox.getSelectionModel().getSelectedItem();  
 dbHandler.deleteTest(nameTest);  
  
 deleteTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CHANGE\_DROP\_TEST*);  
 }  
  
 private void goBack() {  
 goBack\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_TEACHER\_MENU*);  
 }  
  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.TextField;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой изменения списка групп студентов: добавление новой / удаление существующей.  
 \*/*public class ChangeGroupsController {  
  
 WorkWithScene ws = new WorkWithScene();  
  
 @FXML  
 private Button addGroup\_button;  
  
 @FXML  
 private ComboBox<String> choiceGroup\_comboBox;  
  
 @FXML  
 private Button deleteGroup\_button;  
  
 @FXML  
 private Button goBack\_button;  
  
 @FXML  
 private TextField nameGroup\_textField;  
  
 @FXML  
 void initialize() {  
  
 ObservableList<String> groupsList = FXCollections.*observableArrayList*(choiceGroups());  
 choiceGroup\_comboBox.setItems(groupsList);  
  
 choiceGroup\_comboBox.setOnAction(event -> {  
 if (!(choiceGroup\_comboBox.getSelectionModel().getSelectedItem().equals(""))) {  
 deleteGroup\_button.setDisable(false);  
 }  
 });  
  
 nameGroup\_textField.setOnAction(event -> {  
 String str = nameGroup\_textField.getText();  
 if (!(str.equals(""))) {  
 addGroup\_button.setDisable(false);  
 }  
 });  
  
 addGroup\_button.setOnAction(event -> {  
 addGroupToBD();  
 addGroup\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 });  
  
 goBack\_button.setOnAction(event -> {  
 goBack();  
 });  
  
 deleteGroup\_button.setOnAction(event -> {  
 delete();  
 deleteGroup\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 });  
 }  
  
 private void delete() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 String group = choiceGroup\_comboBox.getSelectionModel().getSelectedItem();  
 dbHandler.deleteGroup(group);  
 }  
  
 private void addGroupToBD() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 String group = nameGroup\_textField.getText();  
 dbHandler.addGroup(group);  
 }  
  
 private void goBack() {  
 goBack\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 }  
  
 private ArrayList<String> choiceGroups() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getGroupsList();  
 }  
  
}

package com.example.system\_testing.controller;  
  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.Test;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.TextField;  
import java.util.ArrayList;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой по созданию новых тестов и добавления в них вопросов с ответами.  
 \*/*public class CreateNewTestController {  
  
 WorkWithScene ws = new WorkWithScene();  
  
 @FXML  
 private ComboBox<String> choiceDiscipline\_comboBox;  
  
 @FXML  
 private Button createNewTest\_button;  
  
 @FXML  
 private Button goBack\_button;  
  
 @FXML  
 private TextField nameNewTest\_textField;  
  
 @FXML  
 void initialize() {  
  
 createNewTest\_button.setOnAction(event -> {  
 createNewTest();  
 });  
  
 goBack\_button.setOnAction(event -> {  
 goBack();  
 });  
  
 }  
  
 private void createNewTest() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 Test test = new Test(nameNewTest\_textField.getText(), choiceDiscipline\_comboBox.getSelectionModel().getSelectedItem());  
  
 dbHandler.createTest(test);  
  
 createNewTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CREATE\_QUESTION*, test);  
 }  
  
 private void goBack() {  
 goBack\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_TEACHER\_MENU*);  
 }  
  
 public void setDisciplinesList(ArrayList<String> list) {  
 ObservableList<String> disciplinesList = FXCollections.*observableArrayList*(list);  
 choiceDiscipline\_comboBox.setItems(disciplinesList);  
 }  
}

package com.example.system\_testing.controller;  
  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.Answer;  
import com.example.system\_testing.essences.Question;  
import com.example.system\_testing.essences.Test;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.CheckBox;  
import javafx.scene.control.TextField;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой создания нового вопроса с ответами.  
 \*/*public class CreateQuestionController {  
 WorkWithScene ws = new WorkWithScene();  
 Question question;  
 Test test;  
  
 @FXML  
 private TextField answerFour\_textField;  
  
 @FXML  
 private TextField answerOne\_textField;  
  
 @FXML  
 private TextField answerThree\_textField;  
  
 @FXML  
 private TextField answerTwo\_textField;  
  
 @FXML  
 private Button exit\_button;  
  
 @FXML  
 private CheckBox isTrueFourCheck;  
  
 @FXML  
 private CheckBox isTrueOneCheck;  
  
 @FXML  
 private CheckBox isTrueThreeCheck;  
  
 @FXML  
 private CheckBox isTrueTwoCheck;  
  
 @FXML  
 private TextField nameQuestion\_textField;  
  
 @FXML  
 private Button saveQuestionInTest\_button;  
  
 @FXML  
 void initialize() {  
  
 saveQuestionInTest\_button.setOnAction(event -> {  
 createQuestion();  
 saveQuestionInTest();  
 });  
  
 exit\_button.setOnAction(event -> {  
 exitInMenu();  
 });  
  
 }  
  
 private void createQuestion() {  
  
 question = new Question(nameQuestion\_textField.getText());  
 question.addAnswer(new Answer(answerOne\_textField.getText(), isTrueOneCheck.isSelected()));  
 question.addAnswer(new Answer(answerTwo\_textField.getText(), isTrueTwoCheck.isSelected()));  
 question.addAnswer(new Answer(answerThree\_textField.getText(), isTrueThreeCheck.isSelected()));  
 question.addAnswer(new Answer(answerFour\_textField.getText(), isTrueFourCheck.isSelected()));  
  
 }  
  
 public void saveQuestionInTest() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 dbHandler.addQuestionInBD(test, question);  
  
 saveQuestionInTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CREATE\_QUESTION*, test);  
 }  
  
 public void setNameTest(Test test) {  
 this.test = test;  
 }  
  
 private void exitInMenu() {  
 exit\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_TEACHER\_MENU*);  
 }  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой удаления пользователей из системы.  
 \*/*public class DeleteUserController {  
  
 WorkWithScene ws = new WorkWithScene();  
  
 @FXML  
 private ComboBox<String> choiceUser\_comboBox;  
  
 @FXML  
 private Button delete\_button;  
  
 @FXML  
 private Button goBack\_button;  
  
 @FXML  
 void initialize() {  
  
 ObservableList<String> groupsList = FXCollections.*observableArrayList*(choiceUser());  
 choiceUser\_comboBox.setItems(groupsList);  
  
 goBack\_button.setOnAction(event -> {  
 goBack();  
 });  
  
 delete\_button.setOnAction(event -> {  
 delete();  
 });  
 }  
 private void delete() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 String user = choiceUser\_comboBox.getSelectionModel().getSelectedItem();  
 dbHandler.deleteUser(user);  
 delete\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 }  
  
 private void goBack() {  
 goBack\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 }  
  
 private ArrayList<String> choiceUser() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getUserList();  
 }  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.Test;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.CheckBox;  
import javafx.scene.text.Text;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой прохождения студентом теста.  
 \*/*public class PassTestController {  
  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 WorkWithScene ws = new WorkWithScene();  
 Test test = new Test();  
 int userID;  
  
 Boolean answerOne;  
 Boolean answerTwo;  
 Boolean answerThree;  
 Boolean answerFour;  
  
 String nameTest;  
 int countQuestions = 0, count = 0, countRightAnswer = 0;  
  
 @FXML  
 private Button beginTest\_button;  
  
 @FXML  
 private Button completedTest\_button;  
  
 @FXML  
 private CheckBox isTrueFour\_checkBox;  
  
 @FXML  
 private CheckBox isTrueOne\_checkBox;  
  
 @FXML  
 private CheckBox isTrueThree\_checkBox;  
  
 @FXML  
 private CheckBox isTrueTwo\_checkBox;  
  
 @FXML  
 private Text nameAnswerFour\_text;  
  
 @FXML  
 private Text nameAnswerOne\_text;  
  
 @FXML  
 private Text nameAnswerThree\_text;  
  
 @FXML  
 private Text nameAnswerTwo\_text;  
  
 @FXML  
 private Text nameQuestion\_text;  
  
 @FXML  
 private Text nameTest\_text;  
  
 @FXML  
 private Button nextQuestion\_button;  
  
 @FXML  
 void initialize() {  
  
 }  
  
 @FXML  
 void pushBeginTest(ActionEvent event) {  
  
 beginTest\_button.setVisible(false);  
  
 isTrueOne\_checkBox.setDisable(false);  
 isTrueTwo\_checkBox.setDisable(false);  
 isTrueThree\_checkBox.setDisable(false);  
 isTrueFour\_checkBox.setDisable(false);  
  
 nextQuestion\_button.setDisable(false);  
  
 if (count > 0) {  
  
 nameQuestion\_text.setText(test.getQuestionList().get(count - 1).getNameQuestion());  
 nameAnswerOne\_text.setText(test.getQuestionList().get(count - 1).getAnswerList().get(0).getNameAnswer());  
 nameAnswerTwo\_text.setText(test.getQuestionList().get(count - 1).getAnswerList().get(1).getNameAnswer());  
 nameAnswerThree\_text.setText(test.getQuestionList().get(count - 1).getAnswerList().get(2).getNameAnswer());  
 nameAnswerFour\_text.setText(test.getQuestionList().get(count - 1).getAnswerList().get(3).getNameAnswer());  
 count--;  
  
 }  
 }  
  
 @FXML  
 void pushNextQuestion(ActionEvent event) {  
  
 if (count > 0) {  
  
 answerOne = isTrueOne\_checkBox.isSelected();  
 answerTwo = isTrueTwo\_checkBox.isSelected();  
 answerThree = isTrueThree\_checkBox.isSelected();  
 answerFour = isTrueFour\_checkBox.isSelected();  
  
 calculatingResults(count, answerOne, answerTwo, answerThree, answerFour);  
  
 isTrueOne\_checkBox.setSelected(false);  
 isTrueTwo\_checkBox.setSelected(false);  
 isTrueThree\_checkBox.setSelected(false);  
 isTrueFour\_checkBox.setSelected(false);  
  
 nameQuestion\_text.setText(test.getQuestionList().get(count - 1).getNameQuestion());  
 nameAnswerOne\_text.setText(test.getQuestionList().get(count - 1).getAnswerList().get(0).getNameAnswer());  
 nameAnswerTwo\_text.setText(test.getQuestionList().get(count - 1).getAnswerList().get(1).getNameAnswer());  
 nameAnswerThree\_text.setText(test.getQuestionList().get(count - 1).getAnswerList().get(2).getNameAnswer());  
 nameAnswerFour\_text.setText(test.getQuestionList().get(count - 1).getAnswerList().get(3).getNameAnswer());  
 count--;  
  
 if (count == 0) {  
  
 nextQuestion\_button.setDisable(true);  
 completedTest\_button.setDisable(false);  
  
 completedTest\_button.setOnAction(events -> {  
  
 answerOne = isTrueOne\_checkBox.isSelected();  
 answerTwo = isTrueTwo\_checkBox.isSelected();  
 answerThree = isTrueThree\_checkBox.isSelected();  
 answerFour = isTrueFour\_checkBox.isSelected();  
  
 calculatingResults(count, answerOne, answerTwo, answerThree, answerFour);  
 setAssessment(countRightAnswer, countQuestions);  
  
 completedTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_STUDENT\_MENU*, userID);  
 });  
 }  
 }  
 }  
  
 private void calculatingResults(int count, Boolean answerOne, Boolean answerTwo, Boolean answerThree, Boolean answerFour) {  
 if (test.getQuestionList().get(count).getAnswerList().get(0).getIsTrueAnswer() == answerOne &&  
 test.getQuestionList().get(count).getAnswerList().get(1).getIsTrueAnswer() == answerTwo &&  
 test.getQuestionList().get(count).getAnswerList().get(2).getIsTrueAnswer() == answerThree &&  
 test.getQuestionList().get(count).getAnswerList().get(3).getIsTrueAnswer() == answerFour)  
 {  
 countRightAnswer++;  
 }  
 }  
  
 private void setAssessment(int countRightAnswers, int countQuestion) {  
 double assessmentDouble = ((double) countRightAnswers / (double) countQuestion) \* 5;  
 int assessment = (int) assessmentDouble;  
 System.*out*.println(assessment + " - оценочка");  
 dbHandler.addAssessmentInBD(test.getNameTest(), userID, assessment);  
 }  
  
 public void setNameTestAndUserID(String nameTest, int userID) {  
 this.userID = userID;  
 this.nameTest = nameTest;  
 nameTest\_text.setText(nameTest);  
 test = dbHandler.getTest(nameTest);  
  
 ArrayList<String> questionList = dbHandler.getQuestionsList(nameTest);  
 for (String nameQuestion: questionList  
 ) {  
 test.addQuestion(dbHandler.getQuestion(nameQuestion));  
 countQuestions++;  
 count++;  
 }  
 }  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.Student;  
import com.example.system\_testing.essences.User;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.TextField;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой регистрации студента в системе.  
 \*/*public class RegStudentController {  
 WorkWithScene ws = new WorkWithScene();  
 String group;  
  
 @FXML  
 private TextField fio\_student\_field;  
  
 @FXML  
 private TextField login\_student\_field;  
  
 @FXML  
 private ComboBox<String> numberGroup\_comboBox;  
  
 @FXML  
 private TextField password\_student\_field;  
  
 @FXML  
 private Button regStudentInSystemButton;  
  
 @FXML  
 void initialize() {  
  
 ObservableList<String> groupsList = FXCollections.*observableArrayList*(choiceGroups());  
 numberGroup\_comboBox.setItems(groupsList);  
  
 regStudentInSystemButton.setOnAction(event -> {  
 signUpNewStudent();  
 });  
  
 }  
  
 private ArrayList<String> choiceGroups() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getGroupsList();  
 }  
  
 private void signUpNewStudent() {  
  
 DataBaseHandler dbHandler = new DataBaseHandler();  
  
 String fio = fio\_student\_field.getText();  
 String login = login\_student\_field.getText();  
 String password = password\_student\_field.getText();  
  
 Student student = new Student(fio, group);  
 User user = new User(login, password, "student");  
  
 dbHandler.signUpUser(user);  
 int userID = dbHandler.getUserID(user);  
  
 group = numberGroup\_comboBox.getSelectionModel().getSelectedItem();  
 int groupID = dbHandler.getGroupID(group);  
  
 if (userID >= 0 && groupID >= 0) {  
 dbHandler.signUpStudent(student, userID, groupID);  
 regStudentInSystemButton.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 } else {  
 System.*out*.println("Где-то ошибка!");  
 }  
 }  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.Teacher;  
import com.example.system\_testing.essences.User;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.TextField;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой регистрации преподавателя в системе.  
 \*/*public class RegTeacherController {  
 ArrayList<String> list = new ArrayList<>();  
 WorkWithScene ws = new WorkWithScene();  
  
 @FXML  
 private Button addDiscipline\_button;  
  
 @FXML  
 private ComboBox<String> choiceDiscipline\_comboBox;  
  
 @FXML  
 private TextField fio\_teacher\_field;  
  
 @FXML  
 private TextField login\_teacher\_field;  
  
 @FXML  
 private TextField password\_teacher\_field;  
  
 @FXML  
 private Button regTeacherInSystem\_button;  
  
 @FXML  
 void initialize() {  
  
 ObservableList<String> disciplinesList = FXCollections.*observableArrayList*(choiceDisciplines());  
 choiceDiscipline\_comboBox.setItems(disciplinesList);  
  
 addDiscipline\_button.setOnAction(event -> {  
 list.add(choiceDiscipline\_comboBox.getSelectionModel().getSelectedItem());  
 });  
  
 regTeacherInSystem\_button.setOnAction(event -> {  
 signUpNewTeacher();  
 });  
  
 }  
  
 private ArrayList<String> choiceDisciplines() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getDisciplinesList();  
 }  
  
 private void signUpNewTeacher() {  
  
 DataBaseHandler dbHandler = new DataBaseHandler();  
  
 String fio = fio\_teacher\_field.getText();  
 String login = login\_teacher\_field.getText();  
 String password = password\_teacher\_field.getText();  
  
 Teacher teacher = new Teacher(fio, list);  
 User user = new User(login, password, "teacher");  
  
 dbHandler.signUpUser(user);  
 int userID = dbHandler.getUserID(user);  
 if (userID >= 0) {  
 dbHandler.signUpTeacher(teacher, userID);  
 dbHandler.connectTeacherAndDisciplines(teacher);  
 regTeacherInSystem\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 } else {  
 System.*out*.println("Где-то ошибка!");  
 }  
  
 }  
}

package com.example.system\_testing.controller;  
  
import java.util.ArrayList;  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import com.example.system\_testing.essences.ResultTest;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.scene.control.cell.PropertyValueFactory;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой просмотра результата прохождения теста группой.  
 \*/*public class ShowResultTestController {  
  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 WorkWithScene ws = new WorkWithScene();  
  
 String numberGroup;  
 String nameTest;  
  
 @FXML  
 private TableColumn<ResultTest, Double> averageAssessment\_column;  
  
 @FXML  
 private ComboBox<String> choiceGroup\_comboBox;  
  
 @FXML  
 private ComboBox<String> choiceTest\_comboBox;  
  
 @FXML  
 private Button goBack\_button;  
  
 @FXML  
 private TableColumn<ResultTest, String> nameTest\_column;  
  
 @FXML  
 private TableColumn<ResultTest, String> numGroup\_column;  
  
 @FXML  
 private TableView<ResultTest> showResult\_table;  
  
 @FXML  
 private Button showResult\_button;  
  
 @FXML  
 void pushShowResult(ActionEvent event) {  
 showResult();  
  
 choiceTest\_comboBox.setDisable(true);  
 showResult\_button.setDisable(true);  
 }  
  
 @FXML  
 void initialize() {  
  
 ObservableList<String> groupList = FXCollections.*observableArrayList*(choiceGroup());  
 choiceGroup\_comboBox.setItems(groupList);  
  
 choiceGroup\_comboBox.setOnAction(event -> {  
 numberGroup = choiceGroup\_comboBox.getSelectionModel().getSelectedItem();  
 if (!(numberGroup.equals(""))) {  
 choiceTest\_comboBox.setDisable(false);  
 }  
 });  
  
 ObservableList<String> testsList = FXCollections.*observableArrayList*(choiceTest());  
 choiceTest\_comboBox.setItems(testsList);  
  
 choiceTest\_comboBox.setOnAction(event -> {  
 nameTest = choiceTest\_comboBox.getSelectionModel().getSelectedItem();  
 showResult\_button.setDisable(false);  
 });  
  
 goBack\_button.setOnAction(event -> {  
 goBack();  
 });  
  
 }  
  
 private void goBack() {  
 goBack\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_ADMINISTRATOR\_MENU*);  
 }  
  
 private void showResult() {  
  
 ArrayList<Integer> listStudentID;  
 double averageAssessment = 0;  
 int assessment;  
 int testID;  
 int groupID;  
 int sumAssessment = 0;  
 int countStudentsPassTest = 0;  
  
 groupID = dbHandler.getGroupID(numberGroup);  
  
 testID = dbHandler.getTestID(nameTest);  
 listStudentID = dbHandler.getListStudentID(groupID);  
  
 for (int studentID: listStudentID  
 ) {  
 assessment = dbHandler.getAssessmentBehindTest(studentID, testID);  
 if (assessment >= 0) {  
 countStudentsPassTest++;  
 sumAssessment += assessment;  
 }  
 }  
  
 if (sumAssessment != 0 && countStudentsPassTest != 0) {  
  
 averageAssessment = (double) sumAssessment / (double) countStudentsPassTest;  
  
 ResultTest resultTest = new ResultTest(numberGroup, nameTest, averageAssessment);  
  
 ObservableList<ResultTest> list = FXCollections.*observableArrayList*(resultTest);  
  
 numGroup\_column.setCellValueFactory(new PropertyValueFactory<>("numberGroup"));  
 nameTest\_column.setCellValueFactory(new PropertyValueFactory<>("nameTest"));  
 averageAssessment\_column.setCellValueFactory(new PropertyValueFactory<>("averageAssessment"));  
  
 showResult\_table.setItems(list);  
  
 }  
  
 }  
  
 private ArrayList<String> choiceTest() {  
 return dbHandler.getTestList();  
 }  
  
 private ArrayList<String> choiceGroup() {  
 return dbHandler.getGroupsList();  
 }  
  
}

package com.example.system\_testing.controller;  
  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import java.util.ArrayList;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой меню студента.  
 \*/*public class StudentMenuController {  
  
 WorkWithScene ws = new WorkWithScene();  
 String nameTest;  
 int userID;  
  
 @FXML  
 private ComboBox<String> choiceTest\_comboBox;  
  
 @FXML  
 private Button exitFromSystem\_button;  
  
 @FXML  
 private Button passTest\_button;  
  
 @FXML  
 void selectTest(ActionEvent event) {  
 nameTest = choiceTest\_comboBox.getSelectionModel().getSelectedItem();  
 }  
  
 @FXML  
 void initialize() {  
  
 passTest\_button.setOnAction(event -> {  
 goToWindowPassTest();  
 });  
  
 exitFromSystem\_button.setOnAction(event -> {  
 exitFromSystem();  
 });  
  
 }  
  
 private void exitFromSystem() {  
 exitFromSystem\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_AUTHENTICATION*);  
 }  
  
 private void goToWindowPassTest() {  
 passTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_PASS\_TEST*, nameTest, userID);  
 }  
  
 private ArrayList<String> choiceTest() {  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 return dbHandler.getTestListFromGroup(userID);  
 }  
  
 public void setUserID(int id) {  
 this.userID = id;  
 ObservableList<String> testsList = FXCollections.*observableArrayList*(choiceTest());  
 choiceTest\_comboBox.setItems(testsList);  
 }  
}

package com.example.system\_testing.controller;  
  
import com.example.system\_testing.auxiliary.ConstNameWindows;  
import com.example.system\_testing.auxiliary.WorkWithScene;  
import com.example.system\_testing.database.DataBaseHandler;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import java.util.ArrayList;  
  
*/\*\*  
 \* Класс - контроллер, для управления формой меню преподавателя.  
 \*/*public class TeacherMenuController {  
  
 int userID;  
 WorkWithScene ws = new WorkWithScene();  
  
 @FXML  
 private Button appointTest\_button;  
  
 @FXML  
 private Button createNewTest\_button;  
  
 @FXML  
 private Button exitFromSystem\_button;  
  
 @FXML  
 private Button updateDropTest\_button;  
  
 @FXML  
 void initialize() {  
  
 createNewTest\_button.setOnAction(event -> {  
 goToWindowCreateNewTest();  
 });  
  
 updateDropTest\_button.setOnAction(event -> {  
 goToWindowUpdateDropTest();  
 });  
  
 appointTest\_button.setOnAction(event -> {  
 goToWindowAppointTest();  
 });  
  
 exitFromSystem\_button.setOnAction(event -> {  
 exitFromSystem();  
 });  
  
 }  
  
 private void goToWindowCreateNewTest() {  
 createNewTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CREATE\_NEW\_TEST*, getDisciplinesListFromTeacher(userID));  
  
 }  
 private void goToWindowUpdateDropTest() {  
 updateDropTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_CHANGE\_DROP\_TEST*);  
 }  
  
 private void goToWindowAppointTest() {  
 appointTest\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_APPOINT\_DATE\_TEST*);  
 }  
  
 private void exitFromSystem() {  
 exitFromSystem\_button.getScene().getWindow().hide();  
 ws.getNewWindow(ConstNameWindows.*WINDOW\_AUTHENTICATION*);  
 }  
  
 public void setUserID(int id) {  
 userID = id;  
 }  
  
 public ArrayList<String> getDisciplinesListFromTeacher (int userID) {  
 ArrayList<String> disciplinesList = new ArrayList<>();  
 DataBaseHandler dbHandler = new DataBaseHandler();  
 disciplinesList = dbHandler.getDisciplinesListFromTeacher(disciplinesList, userID);  
 return disciplinesList;  
 }  
}

package com.example.system\_testing.database;  
  
*/\*\*  
 \* Необходимые данные для подключения к БД.  
 \*/*public class Configs {  
 protected String dbHost = "localhost";  
 protected String dbPort = "3306";  
 protected String dbUser = "root";  
 protected String dbPass = "170298fhntV!";  
 protected String dbName = "system\_testing\_db";  
}

package com.example.system\_testing.database;  
  
import com.example.system\_testing.auxiliary.ConstTables;  
import com.example.system\_testing.essences.\*;  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.Date;  
  
*/\*\*  
 \* Класс с методами для соединения и взаимодействия с таблицами базы данных..  
 \*/*public class DataBaseHandler extends Configs {  
 Connection dbConnection;  
  
 */\*\*  
 \* Подключение к БД.  
 \** ***@return*** *- возвращает драйвед для соединения.  
 \** ***@throws*** *ClassNotFoundException - ошибочки.  
 \** ***@throws*** *SQLException - ошибочки.  
 \*/* public Connection getDbConnection()  
 throws ClassNotFoundException, SQLException {  
 String connectionString = "jdbc:mysql://" + dbHost + ":" + dbPort + "/" + dbName;  
  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
  
 dbConnection = DriverManager.*getConnection*(connectionString, dbUser, dbPass);  
  
 return dbConnection;  
 }  
  
 */\*\*  
 \* Добавление дисциплины в БД.  
 \** ***@param*** *discipline - наименование дисциплины.  
 \*/* public void addDiscipline(String discipline) {  
 String insertDB = "INSERT INTO " + ConstTables.*DISCIPLINES\_TABLE* + "(" + ConstTables.*DISCIPLINES\_NAME* + ")" + "VALUES(?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertDB);  
  
 prSt.setString(1, discipline);  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Добавление группы в БД.  
 \** ***@param*** *group - номер группы.  
 \*/* public void addGroup(String group) {  
 String insertDB = "INSERT INTO " + ConstTables.*GROUPS\_TABLE* + "(" + ConstTables.*GROUPS\_NUMBER* + ")" + "VALUES(?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertDB);  
  
 prSt.setString(1, group);  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Добавление вопроса и ответов на него в БД.  
 \** ***@param*** *test - тест.  
 \** ***@param*** *question - вопрос.  
 \*/* public void addQuestionInBD(Test test, Question question) {  
 String insertQuestionDB = "INSERT INTO " + ConstTables.*QUESTIONS\_TABLE* + "(" + ConstTables.*QUESTIONS\_NAME* + ", " +  
 ConstTables.*QUESTIONS\_TESTES\_ID* + ")" + "VALUES(?, ?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertQuestionDB);  
  
 prSt.setString(1, question.getNameQuestion());  
 prSt.setInt(2, getTestID(test.getNameTest()));  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 for (Answer answer: question.getAnswerList()  
 ) {  
 String insertAnswerBD = "INSERT INTO " + ConstTables.*ANSWERS\_TABLE* + "(" + ConstTables.*ANSWERS\_NAME* + ", " +  
 ConstTables.*ANSWERS\_IS\_TRUE* + ", " + ConstTables.*ANSWERS\_QUESTION\_ID* + ")" + "VALUES(?, ?, ?)";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertAnswerBD);  
  
 prSt.setString(1, answer.getNameAnswer());  
 prSt.setBoolean(2, answer.getIsTrueAnswer());  
 prSt.setInt(3, getQuestionID(question.getNameQuestion()));  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 */\*\*  
 \* Запись данных пользователя в БД.  
 \** ***@param*** *user - пользователь.  
 \*/* public void signUpUser(User user) {  
 String insertDB = "INSERT INTO " + ConstTables.*USERS\_TABLE* + "(" + ConstTables.*USERS\_LOGIN* + ", " + ConstTables.*USERS\_PASSWORD* +  
 ", " + ConstTables.*USERS\_ROLE* + ")" + "VALUES(?,?,?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertDB);  
  
 prSt.setString(1, user.getUserLogin());  
 prSt.setString(2, user.getUserPassword());  
 prSt.setString(3, user.getUserRole());  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Запись данных преподавателя в БД.  
 \** ***@param*** *teacher - преподаватель.  
 \** ***@param*** *userID - ID пользователя.  
 \*/* public void signUpTeacher(Teacher teacher, int userID) {  
 String insertDB = "INSERT INTO " + ConstTables.*TEACHERS\_TABLE* + "(" + ConstTables.*TEACHERS\_FIO* + ", "  
 + ConstTables.*TEACHERS\_USER\_ID* + ")" + "VALUES(?,?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertDB);  
  
 prSt.setString(1, teacher.getFio());  
 prSt.setInt(2, userID);  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Запись данных студента в БД.  
 \** ***@param*** *student - студент.  
 \** ***@param*** *userID - ID пользователя.  
 \** ***@param*** *groupID - ID группы.  
 \*/* public void signUpStudent(Student student, int userID, int groupID) {  
 String insertDB = "INSERT INTO " + ConstTables.*STUDENTS\_TABLE* + "(" + ConstTables.*STUDENTS\_FIO* + ", " + ConstTables.*STUDENTS\_USER\_ID* + ", "  
 + ConstTables.*STUDENTS\_GROUPS\_ID* + ")" + "VALUES(?,?,?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertDB);  
  
 prSt.setString(1, student.getFio());  
 prSt.setInt(2, userID);  
 prSt.setInt(3, groupID);  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Запись назначенной даты теста для группы в БД.  
 \** ***@param*** *date - дата теста.  
 \** ***@param*** *nameTest - название теста.  
 \** ***@param*** *numberGroup - название группы.  
 \*/* public void appointDateInDB(String date, String nameTest, String numberGroup) {  
 int testID = getTestID(nameTest);  
 int groupID = getGroupID(numberGroup);  
  
 String insertDB = "INSERT INTO " + ConstTables.*SCHEDULES\_TABLE* + "(" + ConstTables.*SCHEDULES\_DATE* + ", " + ConstTables.*SCHEDULES\_GROUPS\_ID* + ", "  
 + ConstTables.*SCHEDULES\_TESTES\_ID* + ")" + "VALUES(?,?,?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertDB);  
  
 prSt.setString(1, date);  
 prSt.setInt(2, groupID);  
 prSt.setInt(3, testID);  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Запись данных о дисциплинах, которые ведут преподаватели.  
 \** ***@param*** *teacher - преподаватель.  
 \*/* public void connectTeacherAndDisciplines(Teacher teacher) {  
 String insertBD = "INSERT INTO " + ConstTables.*DISCIPLINES\_HAS\_TEACHERS\_TABLE* + "(" + ConstTables.*DISCIPLINES\_HAS\_TEACHERS\_TEACHERS\_ID* + ", "  
 + ConstTables.*DISCIPLINES\_HAS\_TEACHERS\_DISCIPLINES\_ID* + ")" + "VALUES(?,?)";  
  
 try {  
  
  
 for (String discipline: teacher.getDisciplinesList()  
 ) {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertBD);  
 prSt.setInt(1, getTeacherID(teacher));  
 prSt.setInt(2, getDisciplineID(discipline));  
 prSt.executeUpdate();  
 }  
  
  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Запись оценки в БД.  
 \** ***@param*** *nameTest - название теста  
 \** ***@param*** *userID - ID пользователя  
 \** ***@param*** *assessment - оценка за тест.  
 \*/* public void addAssessmentInBD(String nameTest, int userID, int assessment) {  
 int studentID = getStudentID(userID);  
 int testID = getTestID(nameTest);  
 String insertDB = "INSERT INTO " + ConstTables.*ASSESSMENTS\_TABLE* + "(" + ConstTables.*ASSESSMENTS\_ASSESSMENT* + ", "  
 + ConstTables.*ASSESSMENTS\_STUDENT\_ID* + ", " + ConstTables.*ASSESSMENTS\_TESTES\_ID* + ")" + "VALUES(?,?,?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertDB);  
  
 prSt.setInt(1, assessment);  
 prSt.setInt(2, studentID);  
 prSt.setInt(3, testID);  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 */\*\*  
 \* Поиск данных студентов из заданной группы.  
 \** ***@param*** *numberGroup - номер группы.  
 \** ***@return*** *- возвращает результат поиска.  
 \*/* public ResultSet getStudentsFromGroup(String numberGroup) {  
 ResultSet resultSet = null;  
  
 String selectDB = "SELECT \* FROM " + ConstTables.*STUDENTS\_TABLE* +  
 " WHERE " + ConstTables.*STUDENTS\_GROUPS\_ID* + " = " + getGroupID(numberGroup);  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectDB);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 return resultSet;  
 }  
  
 */\*\*  
 \* Получение ID пользователя из БД.  
 \** ***@param*** *user - пользователь.  
 \** ***@return*** *- возвращает целое число.  
 \*/* public int getUserID(User user) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*USERS\_TABLE* +  
 " WHERE " + ConstTables.*USERS\_LOGIN* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setString(1, user.getUserLogin());  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*USERS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID пользователя из БД.  
 \** ***@param*** *login - логин пользователя.  
 \** ***@return*** *- возвращает целое число.  
 \*/* public int getUserID(String login) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*USERS\_TABLE* +  
 " WHERE " + ConstTables.*USERS\_LOGIN* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setString(1, login);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*USERS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID преподавателя из БД.  
 \** ***@param*** *teacher - преподаватель.  
 \** ***@return*** *- возвращает целое число.  
 \*/* public int getTeacherID(Teacher teacher) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*TEACHERS\_TABLE* +  
 " WHERE " + ConstTables.*TEACHERS\_FIO* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setString(1, teacher.getFio());  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*TEACHERS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID преподавателя из БД.  
 \** ***@param*** *userID - ID пользователя.  
 \** ***@return*** *- возвращает ID преподавателя.  
 \*/* public int getTeacherID(int userID) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*TEACHERS\_TABLE* +  
 " WHERE " + ConstTables.*TEACHERS\_USER\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, userID);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*TEACHERS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID студента из БД  
 \** ***@param*** *userID - ID пользователя.  
 \** ***@return*** *- возвращает ID студента.  
 \*/* public int getStudentID(int userID) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*STUDENTS\_TABLE* +  
 " WHERE " + ConstTables.*STUDENTS\_USER\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, userID);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*STUDENTS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID группы из БД  
 \** ***@param*** *userID - ID пользователя.  
 \** ***@return*** *- возвращает ID студента.  
 \*/* public int getGroupID(int userID) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*STUDENTS\_TABLE* +  
 " WHERE " + ConstTables.*STUDENTS\_USER\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, userID);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*SCHEDULES\_GROUPS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID дисциплины из БД.  
 \** ***@param*** *discipline - название дисциплины.  
 \** ***@return*** *- возвращает целое число.  
 \*/* public int getDisciplineID(String discipline) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*DISCIPLINES\_TABLE* +  
 " WHERE " + ConstTables.*DISCIPLINES\_NAME* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setString(1, discipline);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*DISCIPLINES\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID группы из БД.  
 \** ***@param*** *group - номер группы  
 \** ***@return*** *- возвращает целое число.  
 \*/* public int getGroupID(String group) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*GROUPS\_TABLE* +  
 " WHERE " + ConstTables.*GROUPS\_NUMBER* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setString(1, group);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*GROUPS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID теста из БД.  
 \** ***@param*** *nameTest - имя теста.  
 \** ***@return*** *- возвращает ID теста.  
 \*/* public int getTestID(String nameTest) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*TESTS\_TABLE* +  
 " WHERE " + ConstTables.*TESTS\_NAME* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setString(1, nameTest);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*TESTS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение ID вопроса из БД.  
 \** ***@param*** *nameQuestion - наименование вопроса.  
 \** ***@return*** *- возвращает ID вопроса.  
 \*/* public int getQuestionID(String nameQuestion) {  
 int id = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*QUESTIONS\_TABLE* +  
 " WHERE " + ConstTables.*QUESTIONS\_NAME* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setString(1, nameQuestion);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 id = resultSet.getInt(ConstTables.*QUESTIONS\_ID*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return id;  
 }  
  
 */\*\*  
 \* Получение роли пользователя из БД.  
 \** ***@param*** *id - ID пользователя.  
 \** ***@return*** *- возвращает роль пользователя.  
 \*/* public String getUserRole(int id) {  
 String userRole = "";  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*USERS\_TABLE* +  
 " WHERE " + ConstTables.*USERS\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, id);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 userRole = resultSet.getString(ConstTables.*USERS\_ROLE*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return userRole;  
 }  
  
 */\*\*  
 \* Получение списка дисциплин для конкретного преподавателя.  
 \** ***@param*** *disciplinesList - входящий список (пустой)ю  
 \** ***@param*** *userID - ID пользователя.  
 \** ***@return*** *- возвращает список дисциплин.  
 \*/* public ArrayList<String> getDisciplinesListFromTeacher(ArrayList<String> disciplinesList, int userID) {  
  
 int teacherID = getTeacherID(userID);  
 ResultSet resultSetOne = null;  
 ResultSet resultSetTwo = null;  
  
 String selectBD = "SELECT " + ConstTables.*DISCIPLINES\_HAS\_TEACHERS\_DISCIPLINES\_ID* + " FROM " +  
 ConstTables.*DISCIPLINES\_HAS\_TEACHERS\_TABLE* +  
 " WHERE " + ConstTables.*DISCIPLINES\_HAS\_TEACHERS\_TEACHERS\_ID* + " =?";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, teacherID);  
 resultSetOne = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 //assert resultSetOne != null;  
 if (!resultSetOne.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
  
 int disciplineID = resultSetOne.getInt(ConstTables.*DISCIPLINES\_HAS\_TEACHERS\_DISCIPLINES\_ID*);  
 String selectDisciplineBD = "SELECT " + ConstTables.*DISCIPLINES\_NAME* + " FROM " +  
 ConstTables.*DISCIPLINES\_TABLE* +  
 " WHERE " + ConstTables.*DISCIPLINES\_ID* + " =?";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectDisciplineBD);  
 prSt.setInt(1, disciplineID);  
 resultSetTwo = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 //assert resultSetTwo != null;  
 if (!resultSetTwo.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 disciplinesList.add(resultSetTwo.getString(ConstTables.*DISCIPLINES\_NAME*));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return disciplinesList;  
  
 }  
  
 */\*\*  
 \* Получение списка имен тестов по ID тестов из БД.  
 \** ***@param*** *listID - список ID тестов.  
 \** ***@return*** *- возвращает список названий тестов.  
 \*/* public ArrayList<String> getTestList(ArrayList<Integer> listID) {  
 ArrayList<String> nameTestList = new ArrayList<>();  
 ResultSet resultSet = null;  
  
 for (int testID: listID  
 ) {  
 String selectBD = "SELECT \* FROM " + ConstTables.*TESTS\_TABLE* +  
 " WHERE " + ConstTables.*TESTS\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, testID);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 nameTestList.add(resultSet.getString(ConstTables.*TESTS\_NAME*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
  
  
 return nameTestList;  
 }  
  
 */\*\*  
 \* Получение объекта - результат тестирования.  
 \** ***@param*** *numberGroup - номер группы.  
 \** ***@param*** *nameTest - название теста.  
 \** ***@param*** *averageAssessment - средний балл по группе.  
 \** ***@return*** *- возвращает результат тестирования.  
 \*/* public ResultTest getResultTest(String numberGroup, String nameTest, double averageAssessment) {  
 return new ResultTest(numberGroup, nameTest, averageAssessment);  
 }  
  
 */\*\*  
 \* Получение объекта - результат тестирования.  
 \** ***@param*** *groupID - ID группы.  
 \** ***@param*** *testID - ID теста.  
 \** ***@param*** *averageAssessment - средний балл по группе.  
 \** ***@return*** *- возвращает результат тестирования.  
 \*/* public ResultTest getResultTest(int groupID, int testID, double averageAssessment) {  
 String numberGroup = getGroupName(groupID);  
 String nameTest = getNameTest(testID);  
 return new ResultTest(numberGroup, nameTest, averageAssessment);  
 }  
  
 */\*\*  
 \* Получение номера группы по ID группы.  
 \** ***@param*** *groupID - ID группы.  
 \** ***@return*** *- возвращает номер группы.  
 \*/* public String getGroupName(int groupID) {  
 String numberGroup = "";  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*GROUPS\_TABLE* +  
 " WHERE " + ConstTables.*GROUPS\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, groupID);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 numberGroup = resultSet.getString(ConstTables.*GROUPS\_NUMBER*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return numberGroup;  
 }  
  
 */\*\*  
 \* Получение название теста по ID теста.  
 \** ***@param*** *testID - ID теста.  
 \** ***@return*** *возвращает название теста.  
 \*/* public String getNameTest(int testID) {  
 String nameTest = "";  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*TESTS\_TABLE* +  
 " WHERE " + ConstTables.*TESTS\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, testID);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 nameTest = resultSet.getString(ConstTables.*TESTS\_NAME*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return nameTest;  
 }  
  
 */\*\*  
 \* Получение строки из БД с логином и паролем пользователя.  
 \** ***@param*** *user - пользователь.  
 \** ***@return*** *- возвращает строку данных пользователя.  
 \*/* public ResultSet getUser(User user) {  
  
 ResultSet resultSet = null;  
  
 String selectDB = "SELECT \* FROM " + ConstTables.*USERS\_TABLE* + " WHERE " + ConstTables.*USERS\_LOGIN* + "=? AND " + ConstTables.*USERS\_PASSWORD* + "=?";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectDB);  
 prSt.setString(1, user.getUserLogin());  
 prSt.setString(2, user.getUserPassword());  
  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 return resultSet;  
 }  
  
 */\*\*  
 \* Получение списка дисциплин из БД.  
 \** ***@return*** *- возвращает список.  
 \*/* public ArrayList<String> getDisciplinesList () {  
  
 ResultSet resultSet = null;  
 ArrayList<String> list = new ArrayList<>();  
 String selectBD = "SELECT " + ConstTables.*DISCIPLINES\_NAME* + " FROM " + ConstTables.*DISCIPLINES\_TABLE*;  
 PreparedStatement prSt;  
  
 try {  
 prSt = getDbConnection().prepareStatement(selectBD);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 //assert resultSet != null;  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 list.add(resultSet.getString(ConstTables.*DISCIPLINES\_NAME*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return list;  
 }  
  
 */\*\*  
 \* Получение списка групп из БД.  
 \** ***@return*** *- возвращает список.  
 \*/* public ArrayList<String> getGroupsList() {  
  
 ResultSet resultSet = null;  
 ArrayList<String> list = new ArrayList<>();  
 String selectBD = "SELECT " + ConstTables.*GROUPS\_NUMBER* + " FROM " + ConstTables.*GROUPS\_TABLE*;  
 PreparedStatement prSt;  
  
 try {  
 prSt = getDbConnection().prepareStatement(selectBD);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 assert resultSet != null;  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 list.add(resultSet.getString(ConstTables.*GROUPS\_NUMBER*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return list;  
 }  
  
 */\*\*  
 \* Получение списка пользователей из БД.  
 \** ***@return*** *- возвращает список.  
 \*/* public ArrayList<String> getUserList() {  
  
 ResultSet resultSet = null;  
 ArrayList<String> list = new ArrayList<>();  
 String selectBD = "SELECT " + ConstTables.*USERS\_LOGIN* + " FROM " + ConstTables.*USERS\_TABLE*;  
 PreparedStatement prSt;  
  
 try {  
 prSt = getDbConnection().prepareStatement(selectBD);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 assert resultSet != null;  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 list.add(resultSet.getString(ConstTables.*USERS\_LOGIN*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return list;  
 }  
  
 */\*\*  
 \* Получение списка вопросов из БД.  
 \** ***@param*** *nameTest - имя теста  
 \** ***@return*** *- возвращает список вопросов.  
 \*/* public ArrayList<String> getQuestionsList(String nameTest) {  
 int testID = getTestID(nameTest);  
 ResultSet resultSet = null;  
 ArrayList<String> list = new ArrayList<>();  
 String selectBD = "SELECT " + ConstTables.*QUESTIONS\_NAME* + " FROM " + ConstTables.*QUESTIONS\_TABLE* +  
 " WHERE " + ConstTables.*QUESTIONS\_TESTES\_ID* + " = " + testID;  
 PreparedStatement prSt;  
  
 try {  
 prSt = getDbConnection().prepareStatement(selectBD);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 //assert resultSet != null;  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 list.add(resultSet.getString(ConstTables.*QUESTIONS\_NAME*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return list;  
 }  
  
 */\*\*  
 \* Получение списка тестов из БД.  
 \** ***@return*** *- возвращает список тестов.  
 \*/* public ArrayList<String> getTestList() {  
  
 ResultSet resultSet = null;  
 ArrayList<String> list = new ArrayList<>();  
 String selectBD = "SELECT " + ConstTables.*TESTS\_NAME* + " FROM " + ConstTables.*TESTS\_TABLE*;  
 PreparedStatement prSt;  
  
 try {  
 prSt = getDbConnection().prepareStatement(selectBD);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 assert resultSet != null;  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 list.add(resultSet.getString(ConstTables.*TESTS\_NAME*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return list;  
 }  
  
 */\*\*  
 \* Получение списка тестов назначенных студенту, определенной группы.  
 \** ***@param*** *userID - ID пользователя.  
 \** ***@return*** *- возвращает список тестов.  
 \*/* public ArrayList<String> getTestListFromGroup(int userID) {  
 int groupID = getGroupID(userID);  
  
 ResultSet resultSet = null;  
 ArrayList<Integer> list = new ArrayList<>();  
 String selectBD = "SELECT " + ConstTables.*SCHEDULES\_TESTES\_ID* + ", " + ConstTables.*SCHEDULES\_DATE* +  
 " FROM " + ConstTables.*SCHEDULES\_TABLE* +  
 " WHERE " + ConstTables.*SCHEDULES\_GROUPS\_ID* + " = " + groupID;  
 PreparedStatement prSt;  
  
 try {  
 prSt = getDbConnection().prepareStatement(selectBD);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 assert resultSet != null;  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 Date date = resultSet.getDate(ConstTables.*SCHEDULES\_DATE*);  
 Date dateNow = new Date();  
  
 if (date.after(dateNow)) {  
 list.add(resultSet.getInt(ConstTables.*SCHEDULES\_TESTES\_ID*));  
 System.*out*.println("Добавлен");  
 } else {  
 System.*out*.println("не добавлен..");  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println(list);  
 return getTestList(list);  
 }  
  
 */\*\*  
 \* Получение списка студентов заданной группы.  
 \** ***@param*** *groupID - ID группы.  
 \** ***@return*** *- возвращает список ID студентов.  
 \*/* public ArrayList<Integer> getListStudentID(int groupID) {  
 ArrayList<Integer> listStudentID = new ArrayList<>();  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*STUDENTS\_TABLE* +  
 " WHERE " + ConstTables.*STUDENTS\_GROUPS\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, groupID);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 listStudentID.add(resultSet.getInt(ConstTables.*STUDENTS\_ID*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 return listStudentID;  
 }  
  
 */\*\*  
 \* Получение оценки студента за определенный тест.  
 \** ***@param*** *studentID - ID студента.  
 \** ***@param*** *testID - ID теста  
 \** ***@return*** *- возвращает оценку.  
 \*/* public int getAssessmentBehindTest(int studentID, int testID) {  
 int assessment = -1;  
 ResultSet resultSet = null;  
 String selectBD = "SELECT \* FROM " + ConstTables.*ASSESSMENTS\_TABLE* +  
 " WHERE " + ConstTables.*ASSESSMENTS\_TESTES\_ID* + " =?" +  
 " AND " + ConstTables.*ASSESSMENTS\_STUDENT\_ID* + " =?";  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectBD);  
 prSt.setInt(1, testID);  
 prSt.setInt(2, studentID);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 try {  
 assessment = resultSet.getInt(ConstTables.*ASSESSMENTS\_ASSESSMENT*);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 }  
 return assessment;  
 }  
  
 */\*\*  
 \* Поиск результата студента за определённый тест.  
 \** ***@param*** *testID - ID теста.  
 \** ***@param*** *studentID - ID студента.  
 \** ***@return*** *- возвращает результат студента.  
 \*/* public ResultSet getStudentAssessment(int testID, int studentID) {  
 ResultSet resultSet = null;  
  
 String selectDB = "SELECT \* FROM " + ConstTables.*ASSESSMENTS\_TABLE* +  
 " WHERE " + ConstTables.*ASSESSMENTS\_STUDENT\_ID* + "=? AND " + ConstTables.*ASSESSMENTS\_TESTES\_ID* + "=?";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(selectDB);  
 prSt.setInt(1, studentID);  
 prSt.setInt(2, testID);  
  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 return resultSet;  
 }  
  
 */\*\*  
 \* Создание нового теста.  
 \** ***@param*** *test - тест.  
 \*/* public void createTest(Test test) {  
  
 String insertDB = "INSERT INTO " + ConstTables.*TESTS\_TABLE* + "(" + ConstTables.*TESTS\_NAME* + ", " +  
 ConstTables.*TESTS\_DISCIPLINES\_ID* + ")" + "VALUES(?, ?)";  
  
 try {  
 PreparedStatement prSt = getDbConnection().prepareStatement(insertDB);  
  
 prSt.setString(1, test.getNameTest());  
 prSt.setInt(2, getDisciplineID(test.getDiscipline()));  
  
 prSt.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 */\*\*  
 \* Создание Вопроса из БД с заполненными ответами.  
 \** ***@param*** *nameQuestion - имя вопроса.  
 \** ***@return*** *- возвращает вопрос.  
 \*/* public Question getQuestion(String nameQuestion) {  
  
 int questionID = getQuestionID(nameQuestion);  
 Question question = new Question(nameQuestion);  
 ResultSet resultSet = null;  
 ArrayList<String> answerNameList = new ArrayList<>();  
 ArrayList<Boolean> answerIsTrueList = new ArrayList<>();  
 String selectBD = "SELECT " + ConstTables.*ANSWERS\_NAME* + ", " + ConstTables.*ANSWERS\_IS\_TRUE* + " FROM " + ConstTables.*ANSWERS\_TABLE* +  
 " WHERE " + ConstTables.*ANSWERS\_QUESTION\_ID* + " = " + questionID;  
 PreparedStatement prSt;  
  
 try {  
 prSt = getDbConnection().prepareStatement(selectBD);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 assert resultSet != null;  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 answerNameList.add(resultSet.getString(ConstTables.*ANSWERS\_NAME*));  
 answerIsTrueList.add(resultSet.getBoolean(ConstTables.*ANSWERS\_IS\_TRUE*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 question.addAnswer(new Answer(answerNameList.get(0), answerIsTrueList.get(0)));  
 question.addAnswer(new Answer(answerNameList.get(1), answerIsTrueList.get(1)));  
 question.addAnswer(new Answer(answerNameList.get(2), answerIsTrueList.get(2)));  
 question.addAnswer(new Answer(answerNameList.get(3), answerIsTrueList.get(3)));  
  
 return question;  
 }  
  
 */\*\*  
 \* Создание Теста из БД  
 \** ***@param*** *nameTest - название теста  
 \** ***@return*** *- объект Тест.  
 \*/* public Test getTest(String nameTest) {  
 int testID = getTestID(nameTest);  
 Test test = new Test(nameTest);  
 ResultSet resultSet = null;  
 String selectBD = "SELECT " + ConstTables.*TESTS\_NAME* + " FROM " + ConstTables.*TESTS\_TABLE* +  
 " WHERE " + ConstTables.*TESTS\_ID* + " = " + testID;  
 PreparedStatement prSt;  
  
 try {  
 prSt = getDbConnection().prepareStatement(selectBD);  
 resultSet = prSt.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 try {  
 assert resultSet != null;  
 if (!resultSet.next()) {  
 break;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 test = new Test(resultSet.getString(ConstTables.*TESTS\_NAME*));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return test;  
 }  
  
 */\*\*  
 \* Удаление дисциплины из БД.  
 \** ***@param*** *discipline - наименование дисциплины.  
 \*/* public void deleteDiscipline(String discipline) {  
 int disciplineID = getDisciplineID(discipline);  
  
 String deleteBD = "DELETE FROM " + ConstTables.*DISCIPLINES\_TABLE* + " WHERE " +  
 ConstTables.*DISCIPLINES\_ID* + " = " + disciplineID;  
 try {  
 PreparedStatement prSt1 = getDbConnection().prepareStatement(deleteBD);  
 prSt1.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Удаление группы из БД.  
 \** ***@param*** *group - номер группы.  
 \*/* public void deleteGroup(String group) {  
 int groupID = getGroupID(group);  
  
 String deleteBD = "DELETE FROM " + ConstTables.*GROUPS\_TABLE* + " WHERE " +  
 ConstTables.*GROUPS\_ID* + " = " + groupID;  
 try {  
 PreparedStatement prSt1 = getDbConnection().prepareStatement(deleteBD);  
 prSt1.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Удаление вопроса из БД.  
 \** ***@param*** *nameQuestion - название вопроса.  
 \*/* public void deleteQuestion(String nameQuestion) {  
 int questionID = getQuestionID(nameQuestion);  
  
 String deleteBD = "DELETE FROM " + ConstTables.*QUESTIONS\_TABLE* + " WHERE " +  
 ConstTables.*QUESTIONS\_ID* + " = " + questionID;  
 try {  
 PreparedStatement prSt1 = getDbConnection().prepareStatement(deleteBD);  
 prSt1.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Удаление теста из БД.  
 \** ***@param*** *nameTest - название теста  
 \*/* public void deleteTest(String nameTest) {  
 int testID = getTestID(nameTest);  
  
 String deleteBD = "DELETE FROM " + ConstTables.*TESTS\_TABLE* + " WHERE " +  
 ConstTables.*TESTS\_ID* + " = " + testID;  
 try {  
 PreparedStatement prSt1 = getDbConnection().prepareStatement(deleteBD);  
 prSt1.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Удаление пользователя из системы.  
 \** ***@param*** *user - удаляемый пользователь.  
 \*/* public void deleteUser(String user) {  
 int userID = getUserID(user);  
  
 String deleteBD = "DELETE FROM " + ConstTables.*USERS\_TABLE* + " WHERE " +  
 ConstTables.*USERS\_ID* + " = " + userID;  
 try {  
 PreparedStatement prSt1 = getDbConnection().prepareStatement(deleteBD);  
 prSt1.executeUpdate();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}

package com.example.system\_testing.essences;  
  
*/\*\*  
 \* Класс для создания объекта - Ответ.  
 \*/*public class Answer {  
 private String nameAnswer;  
 boolean isTrueAnswer = false;  
  
 public Answer() {  
 }  
  
 public Answer(String nameAnswer, boolean isTrueAnswer) {  
 this.nameAnswer = nameAnswer;  
 this.isTrueAnswer = isTrueAnswer;  
 }  
  
 public String getNameAnswer() {  
 return nameAnswer;  
 }  
  
 public void setNameAnswer(String nameAnswer) {  
 this.nameAnswer = nameAnswer;  
 }  
  
 public boolean getIsTrueAnswer() {  
 return isTrueAnswer;  
 }  
  
 public void setTrueAnswer(boolean trueAnswer) {  
 isTrueAnswer = trueAnswer;  
 }  
}

package com.example.system\_testing.essences;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \* Класс для создания объекта - Вопрос.  
 \*/*public class Question {  
 private String nameQuestion;  
 private ArrayList<Answer> answerList = new ArrayList<>();  
  
 public Question() {  
 }  
  
 public Question(String nameQuestion) {  
 this.nameQuestion = nameQuestion;  
 }  
  
 public String getNameQuestion() {  
 return nameQuestion;  
 }  
  
 public void setNameQuestion(String nameQuestion) {  
 this.nameQuestion = nameQuestion;  
 }  
  
 public ArrayList<Answer> getAnswerList() {  
 return answerList;  
 }  
  
 public void setAnswerList(ArrayList<Answer> answerList) {  
 this.answerList = answerList;  
 }  
  
 public void addAnswer(Answer answer) {  
 this.answerList.add(answer);  
 }  
}

package com.example.system\_testing.essences;  
  
*/\*\*  
 \* Класс для создания объекта - Результат тестирования группы.  
 \*/*public class ResultTest {  
 private String numberGroup;  
 private String nameTest;  
 private double averageAssessment;  
  
 public ResultTest() {  
 }  
  
 public ResultTest(String numberGroup, String nameTest, double averageAssessment) {  
 this.numberGroup = numberGroup;  
 this.nameTest = nameTest;  
 this.averageAssessment = averageAssessment;  
 }  
  
  
  
 public String getNumberGroup() {  
 return numberGroup;  
 }  
  
 public void setNumberGroup(String numberGroup) {  
 this.numberGroup = numberGroup;  
 }  
  
 public String getNameTest() {  
 return nameTest;  
 }  
  
 public void setNameTest(String nameTest) {  
 this.nameTest = nameTest;  
 }  
  
 public double getAverageAssessment() {  
 return averageAssessment;  
 }  
  
 public void setAverageAssessment(int averageAssessment) {  
 this.averageAssessment = averageAssessment;  
 }  
}

package com.example.system\_testing.essences;  
  
*/\*\*  
 \* Класс для создания объекта - Студент.  
 \*/*public class Student {  
 private String fio;  
 private String group;  
  
 public Student() {  
 }  
  
 public Student(String fio, String group) {  
 this.fio = fio;  
 this.group = group;  
 }  
  
 public String getFio() {  
 return fio;  
 }  
  
 public void setFio(String fio) {  
 this.fio = fio;  
 }  
  
 public String getGroup() {  
 return group;  
 }  
  
 public void setGroup(String group) {  
 this.group = group;  
 }  
}

package com.example.system\_testing.essences;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \* Класс для создания объекта - Преподаватель.  
 \*/*public class Teacher {  
 private String fio;  
 private ArrayList<String> disciplinesList = new ArrayList<>();  
  
 public Teacher() {  
 }  
  
 public Teacher(String fio, ArrayList<String> disciplinesList) {  
 this.fio = fio;  
 this.disciplinesList.addAll(disciplinesList);  
 }  
  
 public String getFio() {  
 return fio;  
 }  
  
 public void setFio(String fio) {  
 this.fio = fio;  
 }  
  
 public ArrayList<String> getDisciplinesList() {  
 return disciplinesList;  
 }  
  
 public void setDisciplinesList(ArrayList<String> disciplinesList) {  
 this.disciplinesList = disciplinesList;  
 }  
  
 public void addDisciplines(String disciplines) {  
 this.disciplinesList.add(disciplines);  
 }  
}

package com.example.system\_testing.essences;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \* Класс для создания объекта - Тест.  
 \*/*public class Test {  
 private String nameTest;  
 private String discipline;  
 private ArrayList<Question> questionList = new ArrayList<>();  
  
 public Test() {  
 }  
  
 public Test(String nameTest) {  
 this.nameTest = nameTest;  
 }  
  
 public Test(String nameTest, String discipline) {  
 this.nameTest = nameTest;  
 this.discipline = discipline;  
 }  
  
 public String getNameTest() {  
 return nameTest;  
 }  
  
 public void setNameTest(String nameTest) {  
 this.nameTest = nameTest;  
 }  
  
 public String getDiscipline() {  
 return discipline;  
 }  
  
 public void setDiscipline(String discipline) {  
 this.discipline = discipline;  
 }  
  
 public ArrayList<Question> getQuestionList() {  
 return questionList;  
 }  
  
 public void setQuestionList(ArrayList<Question> questionList) {  
 this.questionList = questionList;  
 }  
  
 public void addQuestion(Question question) {  
 this.questionList.add(question);  
 }  
}

package com.example.system\_testing.essences;  
  
*/\*\*  
 \* Класс для создания объекта - Пользователь.  
 \*/*public class User {  
 private String userLogin;  
 private String userPassword;  
 private String userRole;  
  
 public User(String userName, String userPassword, String userRole) {  
 this.userLogin = userName;  
 this.userPassword = userPassword;  
 this.userRole = userRole;  
 }  
  
 public User() {  
 }  
  
 public String getUserLogin() {  
 return userLogin;  
 }  
  
 public void setUserLogin(String userLogin) {  
 this.userLogin = userLogin;  
 }  
  
 public String getUserPassword() {  
 return userPassword;  
 }  
  
 public void setUserPassword(String userPassword) {  
 this.userPassword = userPassword;  
 }  
  
 public String getUserRole() {  
 return userRole;  
 }  
  
 public void setUserRole(String userRole) {  
 this.userRole = userRole;  
 }  
}