Оленчикова Т.Ю.

**Методические указания и задачи к выполнению лабораторной работы по дисциплине «**Программирование на языке Java для анализа данных**»**

**Лабораторная работа 6 Работа с базами данных**

Цель: Ознакомиться с графическими интерфейсами Java, получить навыки работы с реляционнвми базами данных

**Теоретические материалы**

# [1) Библиотека Swing](https://java-online.ru/libs-swing.xhtml)

# 2) Руководство по JavaFX [Электронный ресурс]. – URL: <https://metanit.com/java/javafx/>

# [3) JDBC - Java DataBase Connectivity](https://java-online.ru/jdbc.xhtml)

Задание.

Необходимо разработать приложение на Java для работы с базой данных. Задание выполняется в 2 этапа:

1) создать экранные формы для ввода запросов к БД и отображения результатов. Иинструкцию по работе см. в приложении 1.

2) реализлвать приложение с графическим интерфейсом для взаимодействия с БД и выполнить запросы. Инструкцию по работе см. в приложении 2.

Используемая СУБД – SQLite/

**Индивидуальные задания для лабораторной работы.**

*Лабораторная работа содержит только 1 вариант*

**Контрольные вопросы**

1. Что такое JDBC?
2. Как осуществляется соединение?
3. Что передается в стороке URL?
4. Что делает класс Statement?
5. Какие есть типы запросов и классы для их выполнения?
6. Для чего нужен класс ResultSet?
7. Как извлекаются данные из ResultSet?
8. Как связаны табличная модель (Table Model) и класс JTable?

**Требования к оформлению отчета.**

Отчет по ЛР состоит из

**а)** Титульный лист: название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания; в) UML-диаграмму классов (создаете в Umlet):

г) Исходный текст программы;

д) Скрины выполнения;

е) Выводы.

**Критерии оценивания.**

За решение задачи вы можете получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки -1 балл; приложение не работоспособно – 0 баллов;

Ответ на контрольный вопрос должен продемонстрировать понимание механизмов Java, за ответ вы можете получить оценку: ответ полный и правильный -1 балл; ответ не удовлетворительный – 0 баллов.

Итого, максимальная оценка - 3 балла

**Внимание!** Полученная оценка автоматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%

**Библиографический список**

1. Программирование на языке Java. Конспект лекций : учебно-методическое пособие / А. В. Гаврилов, С. В. Клименков, Ю. А. Королёва [и др.]. — Санкт-Петербург : НИУ ИТМО, 2019. — С. 5-17, 26-35. — URL: https://e.lanbook.com/book/136549 (дата обращения: 21.08.2023). — Режим доступа: для авториз. пользователей.
2. Пономарчук, Ю. В. Программирование на языке Java : учебное пособие / Ю. В. Пономарчук, И. В. Кузнецов. — Хабаровск : ДВГУПС, 2021. — 103 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/259451 (дата обращения: 21.08.2023). — Режим доступа: для авториз. пользователей.

**Электронные ресурсы**

1. Руководство по языку программирования Java [Электронный ресурс]. – URL: <https://metanit.com/java/tutorial//> – Свободный доступ.
2. Курсы, статьи по Java. – URL: <https://javarush.com/> – Свободный доступ.
3. Учебное пособие по Java [Электронный ресурс]. – URL: <https://www.bestprog.net/ru/sitemap_ru/java/> – Свободный доступ.

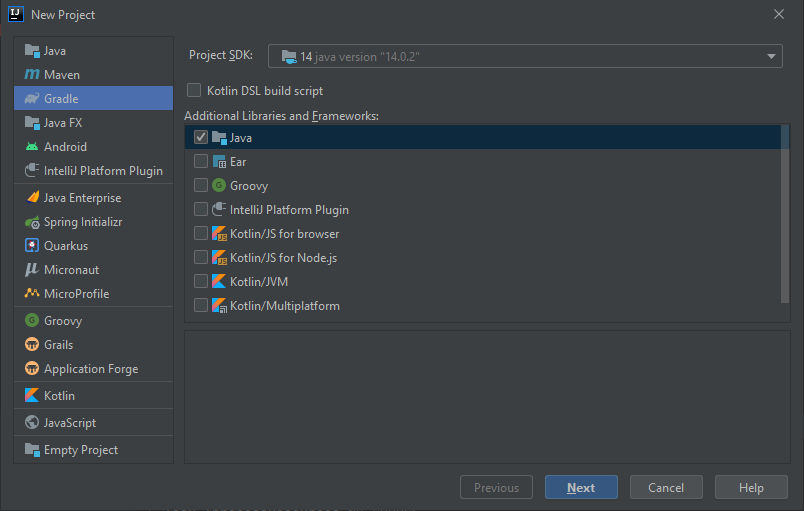
**Приложение 1**

**Тема:** разработка графического интерфейса на языке Java с использованием библиотеки JavaFX.

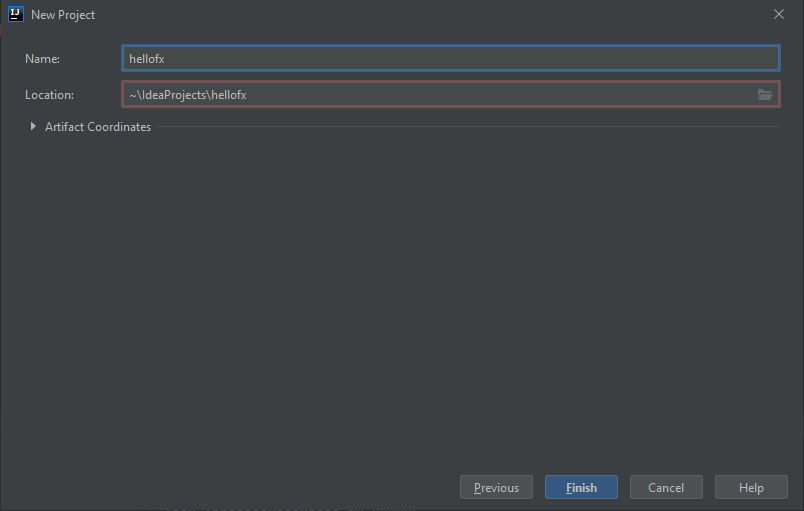
**Задание 1**

1. Откройте *IntellijIDEA*

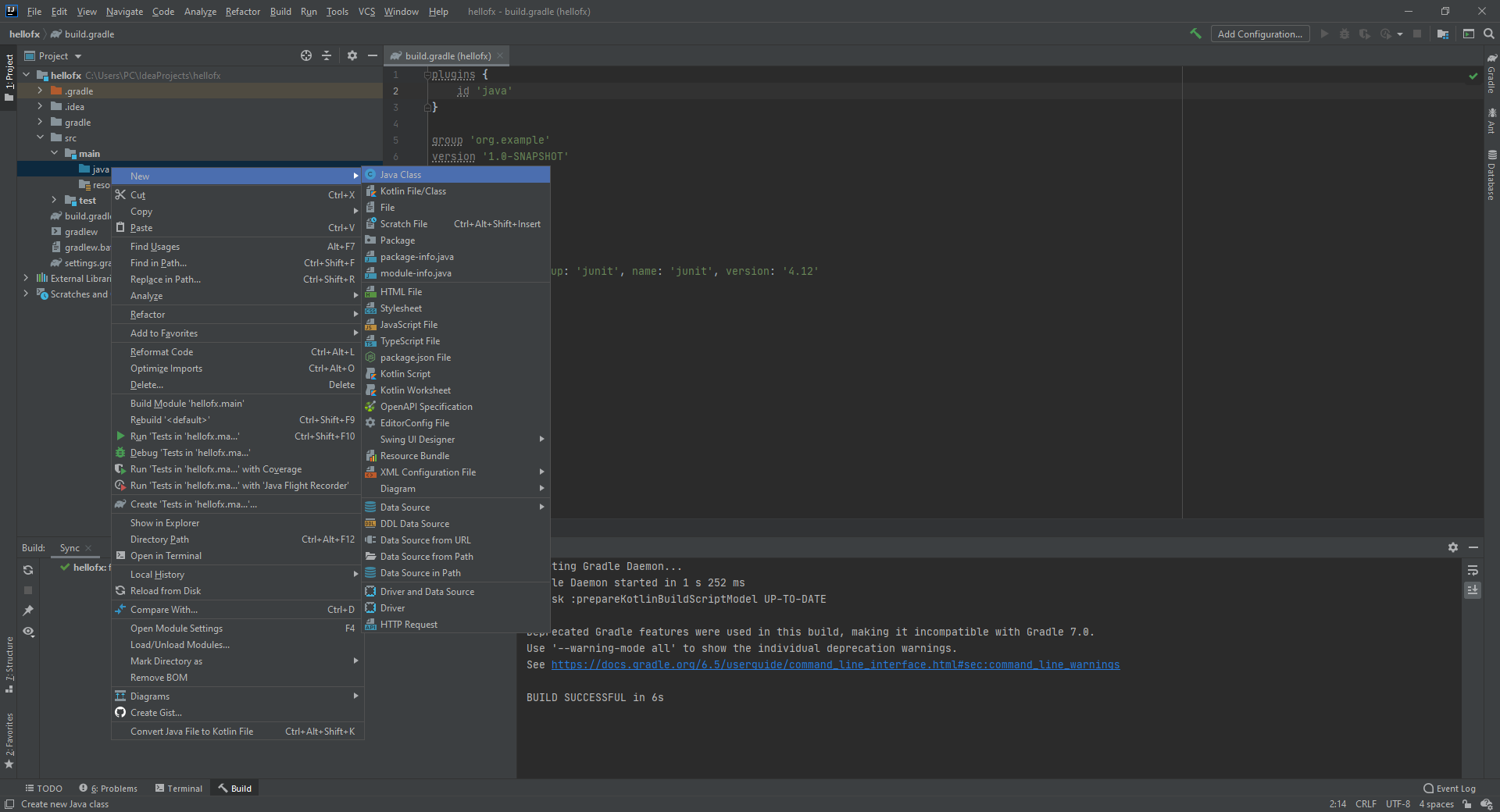
2. Создайте новый *Gradle* проект, нажмите далее:



3. В следующем окне укажите название проекта и выберите его расположения, например:



4. В папке *src/main/java* создайте новый класс, назовите его *MainApp*:



5. Откройте файл *build.gradle* и замените его содержимое на следующее:

plugins {

id 'application'

id 'org.openjfx.javafxplugin' version '0.0.8'

}

repositories {

mavenCentral()

}

dependencies {

}

javafx {

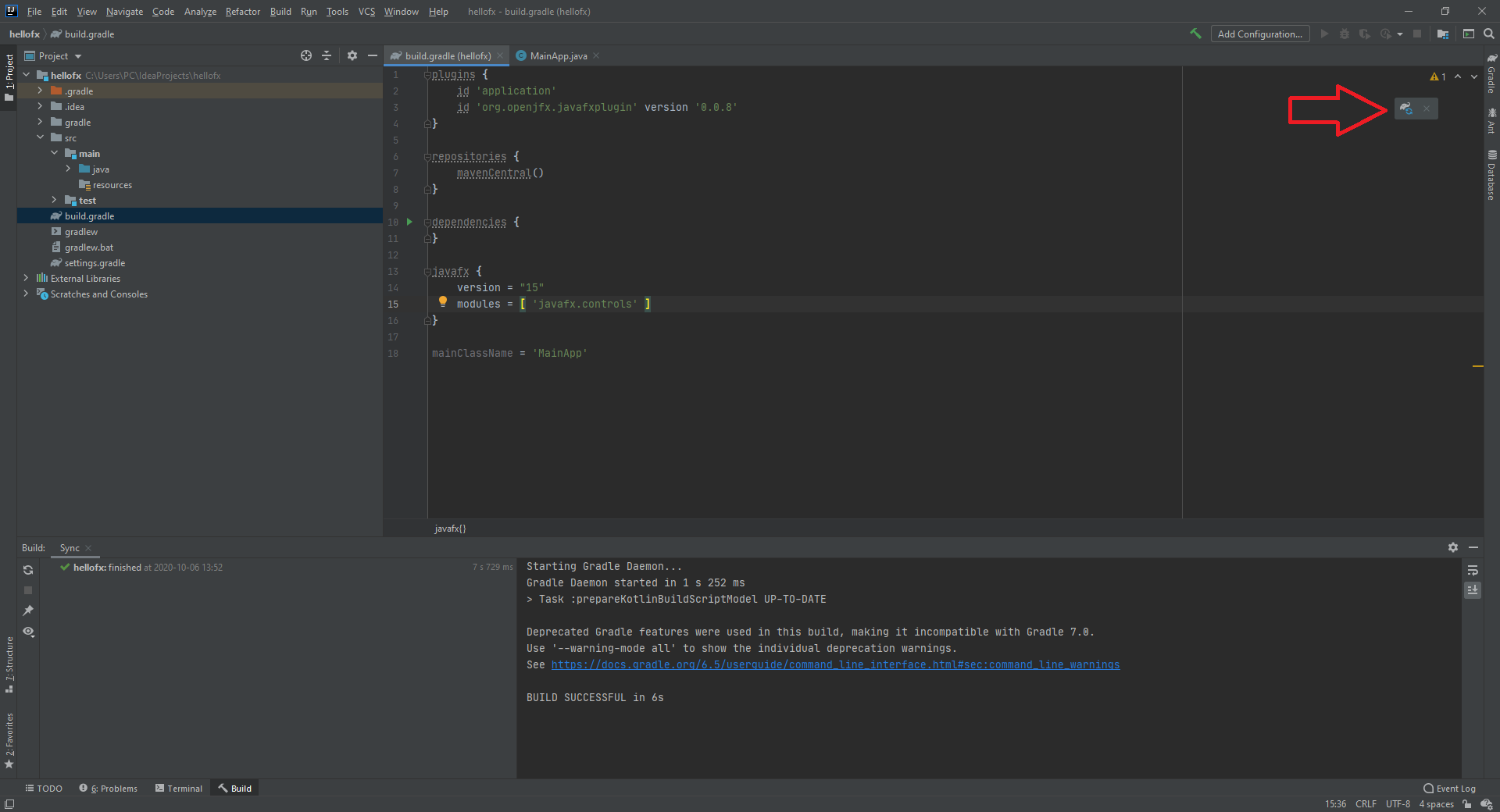
version = "15"

modules = [ 'javafx.controls' ]

}

mainClassName = 'MainApp'

6. После чего обновите конфигурацию *Gradle*:

****

7. Вернитесь к файлу *MainApp.java* и замените его содержимое на:

import javafx.application.Application;

import javafx.stage.Stage;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.BorderPane;

public class MainApp extends Application {

public static void main(String[] args) {

Application.launch(args);

}

@Override

public void start(Stage stage) {

Button btn = new Button("Click");

BorderPane root = new BorderPane(btn);

Scene scene = new Scene(root, 800, 600);

stage.setScene(scene);

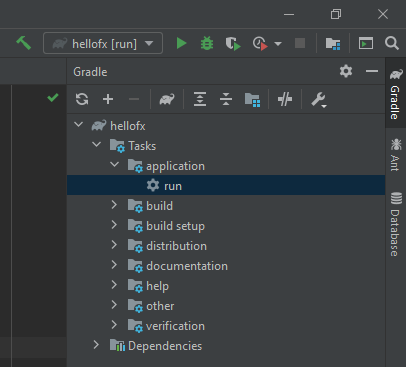
stage.setTitle("HelloFX");

stage.show();

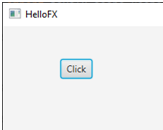
}

}

8. Откройте вкладку *Gradle*, она находится справа. Найдите пункт *application/run* и кликните по нему дважды.



9. После этого должно появиться окно нашего приложения:

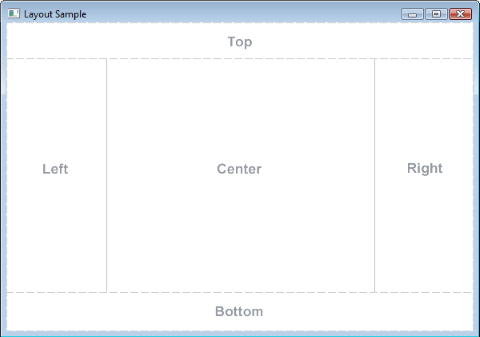
****

10. Разберём код класса *MainApp*.

Выполнение начинается с метода *main*, он запускает внутренние механизмы *JavaFX*, после чего выполняется метод *start*, в нём и происходит построение интерфейса нашего приложения.

В первой строке создаётся кнопка с надписью *Click*. Пока она не выполняет никаких действий.

В следующей строке мы создаём контейнер *BorderPane*. Он позволяет размещать элементы интерфейса по следующей схеме:



Мы передаём кнопку в конструктор *BorderPane*, она автоматически помещается в центр.

В 3 строке создаётся окно приложения. Мы указываем содержимое окна и задаём его размеры.

Затем мы задаём название нашего окна и вызываем метод *show*, чтобы отобразить окно.

11. Измените код, создав контейнер *VBox* и поместив в него несколько элементов:

Label label1 = new Label("Label1");

Label label2 = new Label("Label2");

Label label3 = new Label("Label3");

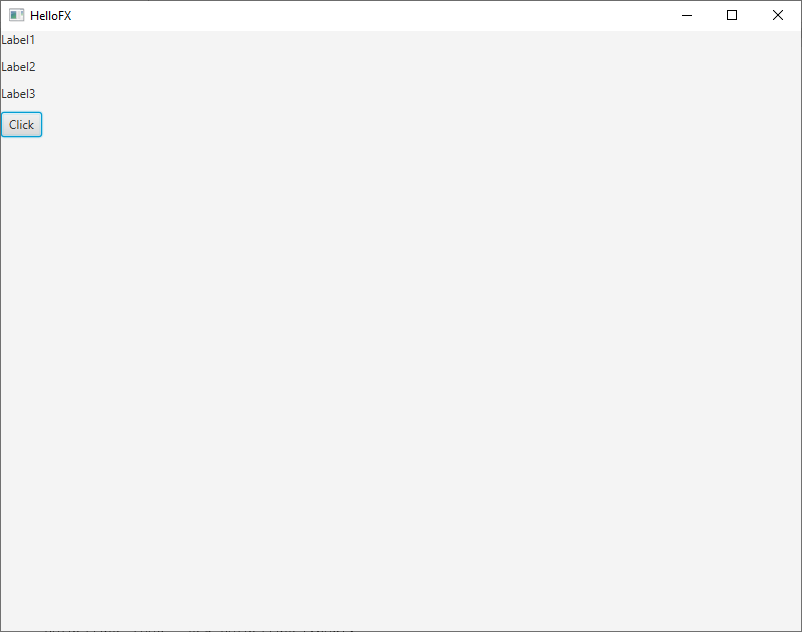
Button btn = new Button("Click");

VBoxvbox = new VBox(10, label1, label2, label3, btn);

BorderPane root = new BorderPane(vbox);

Контейнер *Vbox* позволяет расположить элементы в столбик. Первый параметр задаёт расстояние между элементами.

12. Запустите приложение. Вы должны увидеть следующее:



Контейнер *Vbox* занимает весь экран, поэтому он не центруется. Существует аналогичный контейнер Hbox для горизонтального расположения элементов.

13. Создадим Todo-приложение. Поместите в метод *start* следующий код:

VBoxtodos = new VBox();

ScrollPanescrollPane = new ScrollPane(todos);

TextFieldtextField = new TextField();

Button addButton = new Button("+");

HBox controls = new HBox(8, textField, addButton);

controls.setPadding(new Insets(8));

BorderPane root = new BorderPane();

root.setCenter(scrollPane);

root.setBottom(controls);

Scene scene = new Scene(root, 550, 800);

stage.setScene(scene);

stage.setTitle("HelloFX");

stage.show();

Здесь мы создаём контейнер *todos*, в котором будут лежать задачи; *textField*–поле, для ввода текста новой задачи; *addButton*–кнопка для добавления новой задачи.

14. Добавьте действие при нажатии на кнопку addButton:

addButton.setOnAction((event) -> {

Label text = new Label(textField.getText());

Button removeButton = new Button("x");

HBoxtodo = new HBox(8, text, removeButton);

todo.setPadding(new Insets(8));

todos.getChildren().add(todo);

textField.setText("");

});

При нажатии на кнопку формируется новый блок задачи, он добавляется в список задач, а поле ввода - очищается.

15. Самостоятельно реализуйте удаление задач из списка при нажатии на *removeButton*.

**Приложение 2**

«Взаимодействие Javaс базами данных»

**Цель**: Получить навыки создания методов классов Java выполняющих функционал отправки запросов в базу данных для редактирования таблиц.

**Задание**: Реализовать для приложения, разработанного в рамках приложения 1, функционал для кнопок таким образом, чтобы процессы, исполняемые внутри приложения, применялись к подключенной к приложению базе данных.

В предыдуще задании вами была написана программа, реализующая заполнение списка задач с помощью ввода в текстовом поле и подтверждения в виде addButton, отображения введенных задач в интерфейсе контейнера todos, а также, в рамках самостоятельного изучения, была реализована кнопка удаленияremoveButton, которая удаляет отображенную задачу из списка вместе с удалением соответствующего ему элемента интерфейса.

Однако, если мы хотим, чтобы приложение при закрытии запоминало наш созданный список задач, необходимо где-то хранить наши внесенные данные. Для решения этой задачи, в рамках лабораторной работы, мы научимся работать с содержимым базы данных.

Для работы с базой данных, в Java используется модуль java.sql.В проекте подключим его как java.sql.\*;

В первую очередь, создадим метод, который будет создавать в нашей базе данных таблицу, для хранения записанных пользователем задач. Для создания воспользуемся реализацией запроса SQL:

CREATE TABLE IF NOT EXISTS tasks id integer PRIMARY KEY,description text NOT NULL

Для того, чтобы передать этот запрос в базу данных, сначала зададим запрос в качестве строки, которую в последствии поместим в объект из java.sql типа Statement. Объект типа Statement используется для реализации записанных в них SQL запросов. Реализуем все это в методе createTable() с учетом возможных ошибок:

private static void createTable() {  
String sql= "CREATE TABLE IF NOT EXISTS tasks (\n"  
+ " id integer PRIMARY KEY,\n"  
+ " description text NOT NULL\n"  
+ ");";  
  
try (Statement stmt= *conn*.createStatement()) {  
stmt.execute(sql);  
 } catch (SQLExceptione) {  
System.*out*.println(e.getMessage());  
 }  
}

Следующим этапом реализуем метод addTask, который будет передавать запрос в базу данных каждый раз, когда пользователь будет использовать addButton. В сам же метод, будем передавать значение description в качестве строки, которое будет формироваться из данных введенных в поле, для ввода текста новой задачиtex tField. В результате, с учетом возврата всех возможных ошибок, пойманных в trycatch, получится следующий код:

private static void addTask(String description) {  
String sql= "INSERT INTO tasks (description) VALUES (?)";  
  
try (PreparedStatementstmt= *conn*.prepareStatement(sql, Statement.*RETURN\_GENERATED\_KEYS*)) {  
stmt.setString(1, description);  
intaffectedRows= stmt.executeUpdate();  
  
if (affectedRows== 0) {  
throw new SQLException("Неудалосьдобавитьзадачу");  
 }  
  
try (ResultSetgeneratedKeys= stmt.getGeneratedKeys()) {  
if (generatedKeys.next()) {  
*renderTask*(generatedKeys.getLong(1), description);  
 } else {  
throw new SQLException("Неудалосьполучить id");  
 }  
 }  
 } catch (SQLExceptione) {  
System.*out*.println(e.getMessage());  
 }  
}

Для отображения данных из базы необходимо задать запрос, который будет брать все эти данные и выводить их в интерфейсе приложения.Чтобы отобразить все занесенные строки с кнопками удаления, необходимо реализовать renderTask(), опираясь на механизм создания кнопки с подписью из предыдущей лабораторной работы.

Для этого реализуем метод loadTasks() с перебором по ключам из таблицы:

**privatestaticvoid**loadTasks() {

String sql = "SELECT \* FROM tasks";

**try** (PreparedStatement stmt = conn.prepareStatement(sql)){

ResultSet rs = stmt.executeQuery();

**while** (rs.next()) {

renderTask(rs.getLong("id"), rs.getString("description"));

}

} **catch** (SQLException e) {

System.out.println(e.getMessage());

}

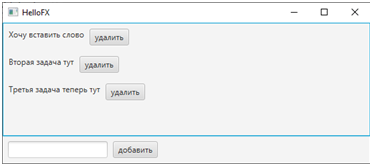
}

Для того чтобы мы смогли выбрать нужную нам базу данных для работы подключим еще java.nio.file.Path и java.nio.file.Paths. Настроим теперь чтобы в основной функции, при запуске программы, у нас подключалась (или если отсутствует, то создавалась) база данных и там была определена таблица с задачами внесенными через интерфейс приложения:

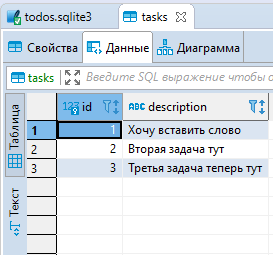
public static void main(String[] args) {  
try {  
Path currentPath= Paths.*get*(System.*getProperty*("user.dir"));  
Path filePath= Paths.*get*(currentPath.toString(), "todos.sqlite3");  
  
String url= "jdbc:sqlite:" + filePath;  
*conn* = DriverManager.*getConnection*(url);  
  
*createTable*();  
*loadTasks*();  
  
Application.*launch*(args);  
 } catch (SQLExceptione) {  
System.*out*.println(e.getMessage());  
 } finally {  
try {  
if (*conn* != null) {  
*conn*.close();  
 }  
 } catch (SQLExceptionex) {  
System.*out*.println(ex.getMessage());  
 }  
 }  
}

А для кнопки addButton добавим событие при нажатии:  
addButton.setOnAction((event) -> {  
*addTask*(textField.getText());  
textField.setText("");  
});

Для того чтобы убедиться в том, что программа работает корректно, заполним через интерфейс несколько задач, которые в последствии будут добавлены в базу todos:



Для того, чтобы проверить содержимое базы данных, можно использовать любое из приложений, однако, в рамках примера, для просмотра данных будет представлен скриншот из программы для просмотра содержимого таблиц баз данных, приложение DBeaver (<https://dbeaver.io/>):



В рамках самостоятельной работы, реализуйте для приложения, сделанного в рамках самостоятельной работы предыдущего задания, событие при нажатии кнопки удаления задач таким образом, чтобы в базе данных также удалялась строка с содержимым этой задачи.