[Введение в Java FX (javarush.ru)](https://javarush.ru/groups/posts/2560-vvedenie-v-java-fx)

**Что такое JavaFX?**

JavaFX — это по сути инструментарий GUI для Java. Здесь будет небольшое отступление, и мы вспомним, что такое **GUI**: **Graphical user interface** — графический интерфейс пользователя — это разновидность пользовательского интерфейса, в котором все элементы (кнопки, меню, пиктограммы, списки) представленные пользователю на дисплее, выполнены в виде картинок, графики. В отличие от интерфейса командной строки, в GUI у пользователя есть произвольный доступ к видимым объектам с помощью устройств ввода. Зачастую элементы интерфейса реализованы в виде метафор и отображают их свойства и назначение для облегчение понимания пользователя. **JavaFX** нацелен на создание игр и настольных приложений на Java. По сути им заменят Swing из-за предложенного нового инструмента GUI для Java. Также, он позволяет нам стилизовать файлы компоновки GUI (XML) и сделать их элегантнее с помощью CSS, подобно тому, как мы привыкли к сетевым приложениям. JavaFX дополнительно работает с интегрированной 3D-графикой, а также аудио, видео и встроенными сетевыми приложениями в единый инструментарий GUI… Он прост в освоении и хорошо оптимизирован. Он поддерживает множество операционных систем, а также Windows, UNIX системы и Mac OS.

**Особенности JavaFX:**

* JavaFX изначально поставляется с большим набором частей графического интерфейса, таких как всякие там кнопки, текстовые поля, таблицы, деревья, меню, диаграммы и т.д., что в свою очередь сэкономит нам вагон времени.
* JavaFX часто использует стили CSS, и мы сможем использовать специальный формат FXML для создания GUI, а не делать это в коде Java. Это облегчает быстрое размещение графического интерфейса пользователя или изменение внешнего вида или композиции без необходимости долго играться в коде Java.
* JavaFX имеет готовые к использованию части диаграммы, поэтому нам не нужно писать их с нуля в любое время, когда вам нужна базовая диаграмма.
* JavaFX дополнительно поставляется с поддержкой 3D графики, которая часто полезна, если мы разрабатываем какую-то игру или подобные приложения.

Давайте немного пройдёмся по основным составляющим нашего окна:

* **Stage** — по сути это окружающее окно, которое используется как начальное полотно и содержит в себе остальные компоненты. У приложения может быть несколько stage, но один такой компонент должен быть в любом случае. По сути Stage является основным контейнером и точкой входа.
* **Scene** — отображает содержание **stage** (прям матрёшка). Каждый stage может содержать несколько компонентов — scene, которые можно между собой переключать. Внутри это реализуется графом объектов, который называется — Scene Graph (где каждый элемент — узел, ещё называемый как **Node**).
* **Node** — это элементы управления, например, кнопки метки, или даже макеты (layout), внутри которых может быть несколько вложенных компонентов. У каждой сцены (scene) может быть один вложенный узел (node), но это может быть макет (layout) с несколькими компонентами. Вложенность может быть многоуровневой, когда макеты содержат другие макеты и обычные компоненты. У каждого такого узла есть свой идентификатор, стиль, эффекты, состояние, обработчики событий.

Итак, давайте двигаться немного в сторону кода. Так как у меня юзается Java 8, мне не нужно подтягивать никакие зависимости, так как JavaFx по дефолту есть в JDK(как и в Java 9,10), но если у нас Java 11+, то нужно пойти в maven repository и стянуть оттуда зависимости.

**JavaFX: примеры использования**

Создаем обычный класс с методом main (точку входа):

**public** **class** AppFX **extends** Application {

**public** **static** **void** main(String[] args) {

Application.launch();

}

@Override

**public** **void** start(Stage primaryStage) **throws** Exception {

pivarySatge.show();

}

}

Тут наш класс наследуется от javafx.application.Application). В мейне вызываем статический метод Application — launch() для запуска нашего окна. Также у нас наша idea будет ругаться, на то что мы не реализовали метод Application — start, что мы в итоге и делаем. Для чего он нужен? А для того, чтобы можно было управлять свойствами (функционалом нашего окна). Для этого у нас используется входящий аргумент primaryStage, у которого мы вызываем метод show, чтобы можно было увидеть запускаемое окно в main.

Давайте немного заполним наш метод start:

**public** **void** start(Stage primaryStage) **throws** Exception {

primaryStage.setTitle("Dogs application");

primaryStage.setWidth(500);

primaryStage.setHeight(400);

InputStream iconStream =

getClass().getResourceAsStream("/images/someImage.png");

Image image = **new** Image(iconStream);

primaryStage.getIcons().add(image);

Button button = **new** Button("WOF WOF ???'");

button.setOnAction(e -< {

Alert alert = **new** Alert(Alert.AlertType.INFORMATION, "WOF WOF WOF!!!");

alert.showAndWait();

});

Scene primaryScene = **new** Scene(button);

primaryStage.setScene(primaryScene);

primaryStage.show();

}

Итак, что мы тут видим? Пробежимся построчно: 2 — задаем название самого окна(stage) 3,4 — задаем его размеры 6,7 — задаем путь читающего потока к файлу (иконке) Введение в Java FX - 58 — создаем файл как объект Image, который связан с реальным файлом потоком передаваемым в конструкторе 9 — задаем иконку в верхнюю панель окна 11 — создаем объект кнопки 13-16 — задаем реакцию при нажатии кнопки 17 — создаем сцену, куда помещаем нашу кнопку 18 — сцену помещаем на наше общее окно 20 — задаем флаг видимости для окна И как результат получаем небольшое окошко, для приветствия наших любимых песелей:

Всё выглядит в разы проще, чем Swing, неправда ли? Но ещё не конец. Полностью писать весь код для отображения приложения, не есть хорошо, нужно его как-то делить, дабы сделать его более понятным (графические составляющие в одни корзинки, логику в — другие). И тут на сцену выходит xml…. О боже мой, xml? Именно. А конкретно — используется его специфичная реализация для JavaFX — FXML, в которой мы определяем графические компоненты приложения и их свойства (там всякие размеры и прочее), а после — связываем с контроллером, который и помогает управлять логикой. Давайте рассмотрим пример такого xml:

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

<?language javascript?>

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

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

<?**import** javafx.scene.layout.VBox?>

<VBox xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml"

id="Dogs application" prefHeight="200" prefWidth="300" alignment="center">

<Label text="Wow wow?"/>

<Button fx:id="mainButton" text="Greeting" onAction="buttonClicked()"/>

<fx:script>

function buttonClicked() {

mainButton.setText("Wow wow wow!!!")

}

</fx:script>

</VBox>

2 — язык сценариев который мы юзаем 4-6 — импортируемые данные 8-9 Vbox — контейнер, который размещает подкомпоненты в одной строке. 11 — выводим некий текст 13 — кнопка при нажатии которой мы юзаем метод описанный в скрипте на 15-18 строке Тут должен быть код вызова данного xml файла в методе start, но сейчас это не столь важно, и мы это опустим (ниже будет пример подтягивания данного файла). Итак, xml — это, конечно, хорошо (да не очень), вручную писать их очень заморочено, разве это не прошлый век?

**Знакомство с JavaFX SceneBuilder**

Именно на этом моменте на сцену выходит (барабанная дробь) — SceneBuilder В JavaFX **Scene Builder** — это инструмент, с помощью которого мы можем конструировать наши окна в виде графического интерфейса и после их сохранять, и эта программа на основании результата будет конструировать xml файлы, которые мы будем подтягивать в нашем приложении.

**Небольшое отступление. JavaFX уроки**

Детали установки я упущу, и подробное изучение данного инструмента тоже. Это темы, которые стоит изучить дополнительно. Поэтому всё же оставлю пару интересных ссылочек на JavaFX уроки: [раз](https://code.makery.ch/ru/library/javafx-tutorial/part1/) (онлайн учебник по JavaFX) и [два](https://o7planning.org/ru/10623/javafx-tutorial-for-beginners) (еще один неплохой туториал). Давайте немного пробежимся по небольшому примеру, который я набросал. В итоге у меня получилось, что-то вроде:

*(такое себе окошко для учёта собак)*

При выборе песеля и нажатии кнопки Delete, собака удаляется из нашего списка. При выборе четырехлапого друга и изменении его полей, а после нажатии кнопки Edit — инфа собачки обновляется. Когда нажимаем кнопку New, вылазит окошко для создания записи новой собаки (для начала её имени): После жмем Save и заполняем в первом окне остальные её поля, а затем жмём кнопку Edit для сохранения. Звучит несложно, верно? Давайте посмотрим, как это будем выглядеть у нас в приложении Java. Для начала, я просто оставлю здесь xml макеты для двух этих окон сгенерированных в SceneBuilder: **Первое(базовое):**

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

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

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

<?**import** javafx.scene.control.SplitPane?>

<?**import** javafx.scene.control.TableColumn?>

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

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

<?**import** javafx.scene.layout.AnchorPane?>

<?**import** javafx.scene.layout.ColumnConstraints?>

<?**import** javafx.scene.layout.GridPane?>

<?**import** javafx.scene.layout.RowConstraints?>

<AnchorPane prefHeight="300.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.tutorial.controller.BaseController">

<children>

<SplitPane dividerPositions="0.29797979797979796" prefHeight="300.0" prefWidth="600.0" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">

<items>

<AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0">

<children>

<TableView fx:id="dogs" layoutX="-2.0" layoutY="-4.0" prefHeight="307.0" prefWidth="190.0" AnchorPane.bottomAnchor="-5.0" AnchorPane.leftAnchor="-2.0" AnchorPane.rightAnchor="-13.0" AnchorPane.topAnchor="-4.0">

<columns>

<TableColumn fx:id="nameList" prefWidth="100.33334350585938" text="Nickname" />

</columns>

<columnResizePolicy>

<TableView fx:constant="CONSTRAINED\_RESIZE\_POLICY" />

</columnResizePolicy>

</TableView>

</children>

</AnchorPane>

<AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0">

<children>

<Label layoutX="49.0" layoutY="25.0" text="Person Details" AnchorPane.leftAnchor="5.0" AnchorPane.topAnchor="5.0" />

<GridPane accessibleText="erreererer" gridLinesVisible="true" layoutX="5.0" layoutY="31.0" AnchorPane.leftAnchor="5.0" AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="31.0">

<columnConstraints>

<ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />

<ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />

</columnConstraints>

<rowConstraints>

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

</rowConstraints>

<children>

<Label prefHeight="17.0" prefWidth="70.0" text="Nickname" />

<Label text="Breed" GridPane.rowIndex="1" />

<Label text="Age" GridPane.rowIndex="2" />

<Label text="City" GridPane.rowIndex="3" />

<Label text="Level of training" GridPane.rowIndex="4" />

<TextField fx:id="breed" GridPane.columnIndex="1" GridPane.rowIndex="1" />

<TextField fx:id="age" GridPane.columnIndex="1" GridPane.rowIndex="2" />

<TextField fx:id="city" GridPane.columnIndex="1" GridPane.rowIndex="3" />

<TextField fx:id="levelOfTraining" GridPane.columnIndex="1" GridPane.rowIndex="4" />

<TextField fx:id="name" GridPane.columnIndex="1" />

</children>

</GridPane>

<Button layoutX="251.0" layoutY="259.0" mnemonicParsing="false" onAction="#create" text="New" AnchorPane.bottomAnchor="10.0" AnchorPane.leftAnchor="230.0" AnchorPane.rightAnchor="130.0" AnchorPane.topAnchor="260.0" />

<Button layoutX="316.0" layoutY="262.0" mnemonicParsing="false" onAction="#edit" text="Edit" AnchorPane.bottomAnchor="10.0" AnchorPane.leftAnchor="290.0" AnchorPane.rightAnchor="70.0" AnchorPane.topAnchor="260.0" />

<Button layoutX="360.0" layoutY="262.0" mnemonicParsing="false" onAction="#delete" text="Delete" AnchorPane.bottomAnchor="10.0" AnchorPane.leftAnchor="350.0" AnchorPane.rightAnchor="10.0" AnchorPane.topAnchor="260.0" />

</children>

</AnchorPane>

</items>

</SplitPane>

</children>

</AnchorPane>

**Второе**

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

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

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

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

<?**import** javafx.scene.layout.AnchorPane?>

<?**import** javafx.scene.layout.ColumnConstraints?>

<?**import** javafx.scene.layout.GridPane?>

<?**import** javafx.scene.layout.RowConstraints?>

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

<AnchorPane prefHeight="200.0" prefWidth="300.0" xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.tutorial.controller.NewDogController">

<children>

<GridPane layoutX="31.0" layoutY="25.0" prefHeight="122.0" prefWidth="412.0">

<columnConstraints>

<ColumnConstraints hgrow="SOMETIMES" maxWidth="185.0" minWidth="10.0" prefWidth="149.0" />

<ColumnConstraints hgrow="SOMETIMES" maxWidth="173.0" minWidth="10.0" prefWidth="146.0" />

</columnConstraints>

<rowConstraints>

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />

</rowConstraints>

<children>

<Label prefHeight="48.0" prefWidth="178.0" text="Please, write name:">

<font>

<Font size="20.0" />

</font>

</Label>

<TextField fx:id="nickName" prefHeight="36.0" prefWidth="173.0" GridPane.columnIndex="1" />

</children>

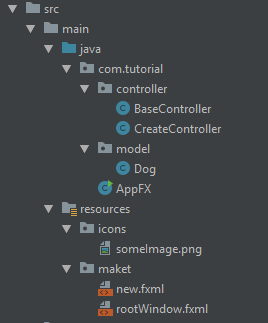
</GridPane>

<Button layoutX="222.0" layoutY="149.0" mnemonicParsing="false" onAction="#ok" prefHeight="37.0" prefWidth="95.0" text="Save" />

<Button layoutX="325.0" layoutY="149.0" mnemonicParsing="false" onAction="#cansel" prefHeight="37.0" prefWidth="95.0" text="Cansel" />

</children>

</AnchorPane>

Как выглядит структура папок: Как видим, ничего особенного, есть контроллеры, представляющие определенные окошки, есть модели представляющие наши данные. Давайте взглянем на класс запускающий приложение (реализация Application): @Data

**public** **class** AppFX **extends** Application {

**private** Stage primaryStage;

**private** AnchorPane rootLayout;

**private** ObservableList listDog = FXCollections.observableArrayList();

**public** AppFX() {

listDog.add(**new** Dog("Fluffy", "Pug", 8, "Odessa", 2));

listDog.add(**new** Dog("Archie", "Poodle", 3, "Lviv", 6));

listDog.add(**new** Dog("Willie", "Bulldog", 5, "Kiev", 4));

listDog.add(**new** Dog("Hector", "Shepherd", 9, "Minsk", 6));

listDog.add(**new** Dog("Duncan", "Dachshund", 1, "Hogwarts", 9));

}

Тут мы видим конструктор, который будет заполнять наши начальные данные (которые храним в специальном листе — ObservableList).

**public** **static** **void** main(String[] args) {

Application.launch();

}

@Override

**public** **void** start(Stage primaryStage) **throws** Exception {

**this**.primaryStage = primaryStage;

**this**.primaryStage.setTitle("Dogs application");

showBaseWindow();

}

Ничего особенного — main и реализация start(), запускающая приложение:

**public** **void** showBaseWindow() {

**try** {

FXMLLoader loader = **new** FXMLLoader();

loader.setLocation(AppFX.**class**.getResource("/maket/rootWindow.fxml"));

rootLayout = loader.load();

Scene scene = **new** Scene(rootLayout);

primaryStage.setScene(scene);

InputStream iconStream = getClass().getResourceAsStream("/icons/someImage.png");

Image image = **new** Image(iconStream);

primaryStage.getIcons().add(image);

BaseController controller = loader.getController();

controller.setAppFX(**this**);

primaryStage.show();

} **catch** (IOException e) {

e.printStackTrace();

}

}

Итак, тут мы видим метод, который мы собственно и запускаем в start(), а именно — задающий настройки нашего базового окна. Таких как на xml макете в ресурсах: задание ему иконки, связывание его с конкретным контроллером, и задание контроллеру ссылки на this класс)

**public** **void** showCreateWindow(Dog dog) {

**try** {

FXMLLoader loader = **new** FXMLLoader();

loader.setLocation(AppFX.**class**.getResource("/maket/new.fxml"));

AnchorPane page = loader.load();

Stage dialogStage = **new** Stage();

dialogStage.setTitle("Wow Wow Wow");

dialogStage.initModality(Modality.WINDOW\_MODAL);

dialogStage.initOwner(primaryStage);

dialogStage.setScene(**new** Scene(page));

CreateController controller = loader.getController();

controller.setDialogStage(dialogStage);

controller.setDog(dog);

dialogStage.showAndWait();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

Тут мы видим метод, который ответственен за появление второго окна — окна создания новой записи (имени новой собаки). Также задаем контроллер, xml макет, stage и прочее… Следующим рассмотренным классом у нас будет модель, представляющая нашу собаку (инфу о ней): @Data

**public** **class** Dog {

**private** StringProperty name;

**private** StringProperty breed;

**private** IntegerProperty age;

**private** StringProperty city;

**private** IntegerProperty levelOfTraining;

**public** Dog(String name, String breed, **int** age, String city, **int** levelOfTraining) {

**this**.name = **new** SimpleStringProperty(name);

**this**.breed = **new** SimpleStringProperty(breed);

**this**.age = **new** SimpleIntegerProperty(age);

**this**.city = **new** SimpleStringProperty(city);

**this**.levelOfTraining = **new** SimpleIntegerProperty(levelOfTraining);

}

**public** Dog() {

name = **new** SimpleStringProperty();

breed = **null**;

age = **null**;

city = **null**;

levelOfTraining = **null**;

}

}

Тут мы видим два конструктора.Один — почти обычный со всеми аргументами (почти, потому что мы юзаем специальные FX оболочки простых типов) и конструктор без аргументов: его мы используем при создании новой собаки, у которой по началу есть только имя. Контроллер для базового окна: @Data

**public** **class** BaseController {

@FXML

**private** TableView dogs;

@FXML

**private** TableColumn nameList;

@FXML

**private** TextField name;

@FXML

**private** TextField breed;

@FXML

**private** TextField age;

@FXML

**private** TextField city;

@FXML

**private** TextField levelOfTraining;

**private** AppFX appFX;

Тут мы видим наши поля объекта, но в формате TextField. Это формат, который представляет поле для ввода текста. @FXML — аннотация предназначенная для связывания кода Java и соответствующего объекта нашего макета (кнопки, поля или ещё чего-то).

@FXML

**private** **void** initialize() {

nameList.setCellValueFactory(

cellData -> cellData.getValue().getName());

dogs.getSelectionModel().selectedItemProperty().addListener(

(observable, oldValue, newValue) -> showDogsInformation(newValue));

}

Тут мы видим метод для вывода имён собак, справа в списке (его аннотация @FXML связывает с компонентом макета JavaFX TableView).

**public** **void** setAppFX(AppFX appFX) {

**this**.appFX = appFX;

dogs.setItems(appFX.getListDog());

}

**private** **void** showDogsInformation(Dog dog) {

**if** (dog != **null**) {

name.setText(dog.getName() != **null** ? dog.getName().getValue() : **null**);

breed.setText(dog.getBreed() != **null** ? dog.getBreed().getValue() : **null**);

age.setText(dog.getAge() != **null** ? String.valueOf(dog.getAge().get()) : **null**);

city.setText(dog.getCity() != **null** ? dog.getCity().getValue() : **null**);

levelOfTraining.setText(dog.getLevelOfTraining() != **null** ? String.valueOf(dog.getLevelOfTraining().get()) : **null**);

} **else** {

name.setText("");

breed.setText("");

age.setText("");

city.setText("");

levelOfTraining.setText("");

}

}

В первом методе мы видим задание внутренней ссылки на класс, реализующий Application (для того, чтобы можно было дёрнуть его метод для вызова второго окна), и задание начального списка для отображения. Второй же проверяет, есть ли определенные данные текущей собаки, и на основании этого задаёт текстовые поля:

@FXML

**private** **void** delete() {

**int** selectedIndex = dogs.getSelectionModel().getSelectedIndex();

dogs.getItems().remove(selectedIndex);

}

@FXML

**private** **void** edit() {

**int** selectedIndex = dogs.getSelectionModel().getSelectedIndex();

dogs.getItems().set(selectedIndex, **new** Dog(name.getText(), breed.getText(), Integer.valueOf(age.getText()), city.getText(), Integer.valueOf(levelOfTraining.getText())));

}

@FXML

**private** **void** create() {

Dog someDog = **new** Dog();

appFX.showCreateWindow(someDog);

**if** (someDog.getName() != **null** && !someDog.getName().getValue().isEmpty()) {

appFX.getListDog().add(someDog);

}

}

}

Тут мы видим три метода, базового окна, связанных с кнопками: Введение в Java FX - 12

* **delete** — по индексу удаляем выбранную(выделенную) собаку;
* **edit** — создаем новую собаку с переданными данными, и задаем ее вместо той которая была до этого;
* **create** — создаем новую собаку и дергаем метод вызова окна создания, передав новый объект, и после закрытия которого если имя не null, то сохраняем нового питомца.

Двигаем дальше, контроллер окна для создания собаки: @Data

**public** **class** CreateController {

**private** Stage dialogStage;

**private** Dog dog;

@FXML

**private** TextField nickName;

@FXML

**private** **void** ok() {

**if** (nickName != **null** && !nickName.getText().isEmpty()) {

dog.setName(**new** SimpleStringProperty(nickName.getText()));

dialogStage.close();

}

}

@FXML

**private** **void** cansel() {

dialogStage.close();

}

}

Тут мы видим связь с текстовым полем в окне, обработки кнопок Save и Cancel, которые так или иначе закрывают окно.