УПРАВЛЕНИЕ ОБРАЗОВАНИЯ АДМИНИСТРАЦИИ НИЖНЕЛОМОВСКОГО РАЙОНА
МУНИЦИПАЛЬНОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
«СРЕДНЯЯ ШКОЛА №4 ГОРОДА НИЖНИЙ ЛОМОВ»
(МБОУ «СШ №4 Г. НИЖНИЙ ЛОМОВ»)

ул. Крылова, д. 6, г. Нижний Ломов, 442151
телефон 4-70-68, E-mail: nlomov4@list.ru
ОГРН 1025800977674, ИНН 5827008319

Научно-практическая конференция «Старт в науку»

Исследовательская работа

«Создание приложения для операционных систем на базе Microsoft Windows средствами языка Java»

Подготовил учащийся 11 «Б»
класса МБОУ «СШ №4 г.
Нижний Ломов»
Акмашев С.

Руководитель Саблина Н.Н.

Февраль, 2020 год

Содержание:

Введение

Любой, кто имеет дело с разработкой приложений на Android, ответит вам, что самый популярный язык программирования в их сфере—это Java. Данный язык является официальным языком разработки на Android, то есть он имеет наибольшую поддержку со стороны Google и большинство приложений в Google Play написаны именно на нем.

Но мало кто знает, что на этом же языке можно писать и приложения для Microsoft Windows.

Цель работы: исследовать возможности языка Java для создания приложения для Microsoft Windows пользователем с начальным уровнем знаний в программировании

Задачи:

- рассмотреть особенности популярности языка Java;
- рассмотреть различные среды для разработки приложения на языке Java
- изучить среду программирования для создания приложения;
- создать собственное приложение

Объект исследования: язык программирования Java

Предмет исследования: среда разработки IntelliJ IDEA.

Гипотеза: пользователь с начальным уровнем знаний в программировании может научиться создавать приложения на языке Java, проявляя своё воображение и творческие способности.

Методы исследования: поисковый, сравнение, сопоставление, анализ.

Ожидаемый результат:

- высокий интерес к среде программирования Java среди обучающихся из младших параллелей
- мотивация на исследовательский аспект изучения литературы

Практическая значимость работы связана с возможностью углубленного, а не поверхностного изучения литературы по программированию, позволяющего продуктивно использовать приобретенные навыки в выборе будущей специальности.

Глава 1.Особенности популярности языка программирования Java

1.1 Краткая история языка Java

По результатам ежегодного отчёта State of the Octoverse, который выпускает Github, язык программирования Java в 2019 году занимает третье место в списке самых популярных.

Java разработала компания Sun Microsystems в начале 90-х годов XX века. Ведущую роль в создании языка сыграл канадский инженер Джеймс Гослинг (James Gosling). На ранних этапах разработки язык назывался Oak. Затем его переименовали в честь сорта кофе Java. Связь языка с напитком отражается в логотипе.

Джеймс Гослинг и его единомышленники хотели создать язык с си-подобным синтаксисом. В то же время он должен быть более простым по сравнению с C/C++. Создатели планировали использовать Java для программирования бытовой электроники. Однако практически сразу после выпуска версии 1.0 в 1995 язык стали использовать разработчики серверного и клиентского ПО.

Название языка читается как «джава». Однако русскоязычные пользователи в разговорной и даже в письменной речи иногда говорят «язык программирования ява». Это один из примеров **использования сленга**.

В 2010 году компанию Sun Microsystems купила Oracle. После этого Джеймс Гослинг перешёл в Google, откуда тоже вскоре уволился.

1.2 Особенности языка Java

Java — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией.

Java является объектно-ориентированным языком, относится к языкам программирования с сильной типизацией.

Создатели реализовали принцип WORA: write once, run anywhere или «пиши один раз, запускай везде». Это значит, что написанное на Java приложение можно запустить на любой платформе, если на ней установлена среда исполнения Java (JRE, Java Runtime Environment).

Эта задача решается благодаря компиляции написанного на Java кода в байт-код. Этот формат исполняет JVM или виртуальная машина Java. JVM — часть среды исполнения Java (JRE). Виртуальная машина не зависит от платформы.

В Java реализован механизм управления памятью, который называется сборщиком мусора или garbage collector. Разработчик создаёт объекты, а JRE с помощью сборщика мусора очищает память, когда объекты перестают использоваться. Объясняет эксперт Никита Липский: «Есть такое понятие — циклический мусор. Внутри цикла на все объекты есть ссылки, однако garbage collector в Java удалит его, если объекты не могут использоваться из программы».

Как отмечалось выше, синтаксис языка Java похож на синтаксис других си-подобных языков. Вот его некоторые особенности:

- чувствительность к регистру — идентификаторы User и user в Java представляют собой разные сущности;
- для именования методов используется lowerCamelCase. Если название метода состоит из одного слова, оно должно начинаться со строчной буквы. Пример: firstMethodName();
- для именования классов используется UpperCamelCase. Если название состоит из одного слова, оно должно начинаться с прописной буквы. Пример: FirstClassName.
- название файлов программы должно точно совпадать с названием класса с учётом чувствительности к регистру. Например, если класс называется FirstClassName, файл должен называться FirstClassName.java;
- идентификаторы всегда начинаются с буквы (A-Z, a-z), знака $ или нижнего подчёркивания _;

Ближе познакомиться с синтаксисом Java можно на бесплатном ресурсе Code Basics, где есть вводный курс.

1.3. Что пишут на языке Java: сферы применения

Выше отмечено, что Java относится к языкам программирования общего назначения. Новичкам интересно знать, что конкретно пишут Java-программисты, чтобы определиться с выбором языка.

По данным компании Oracle, программы на Java запускаются на 3 млрд девайсов. Именно такое сообщение можно увидеть в окне установки.

Маркетинговое сообщение в окне установки Java

Это маркетинговое сообщение сложно проверить. Тем не менее Java широко используется и входит в число самых востребованных языков, это не вызывает сомнения.

Например, подавляющее большинство крупных компаний так или иначе используют Java.

Области применения Java

- Очень много серверных приложений для корпораций написаны на этом языке. Например, речь идёт о программах для финансовых организаций, которые обеспечивают проведение транзакций, фиксацию торговых операций.
- На Java написано много веб-приложений. Популярные фреймворки, в том числе Spring, Stuts, JSP, используются для создания разных приложений в вебе: от ecommerce-проектов до крупных

порталов, от образовательных платформ до правительственных ресурсов.

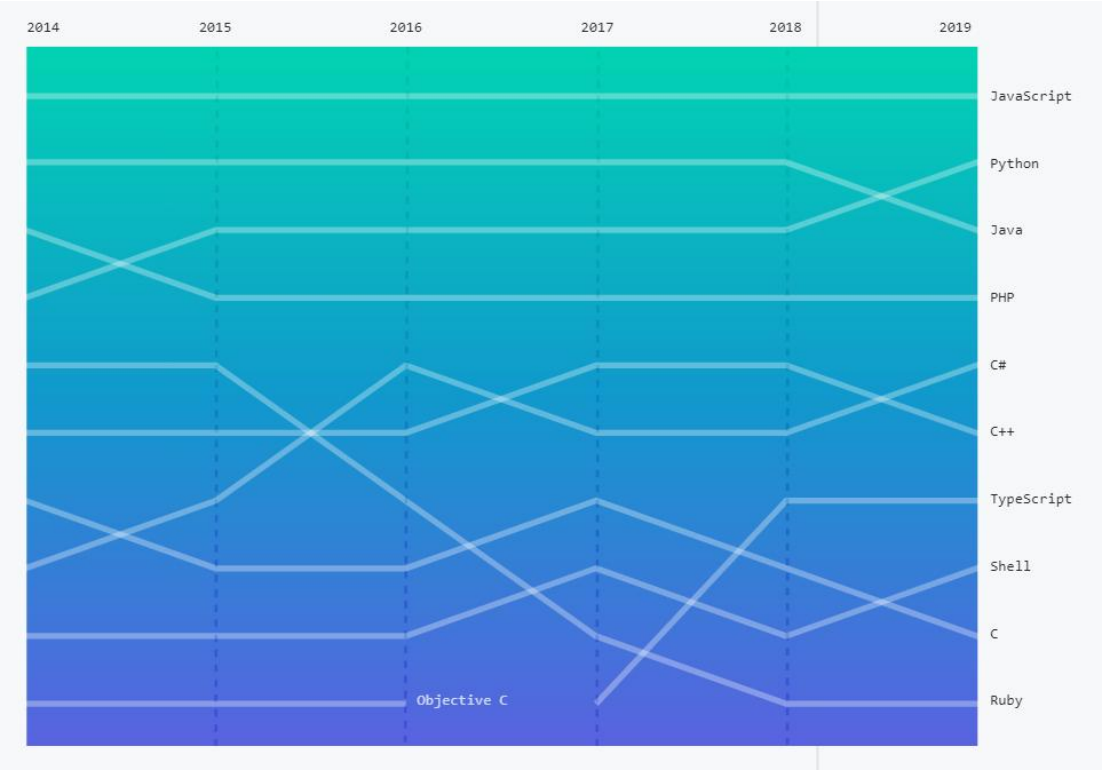
- Популярная компьютерная игра Minecraft написана на Java.
- Мобильная разработка — ещё одна область использования Java. На этом языке пишут приложения для устройств, работающих под управлением ОС Android.
- На Java создают клиентские приложения. Простой и близкий разработчикам пример: IDE NetBeans написано на «джаве».
- Также Java применяется для работы с Big Data, разработки программ для научных целей, например, обработки естественных языков, программирования приборов — от бытовых девайсов до промышленных установок.

То есть на Java можно писать разные типы приложений: веб, мобильный и десктопный софт, игры и так далее. Традиционно у этого языка сильные позиции в промышленном программировании, в сегменте крупных компаний (т.н. энтерпрайз).

Такми образом можно сделать вывод о том, что Java — язык программирования общего назначения. Имеет си-подобный синтаксис. Используется для создания приложений в разных областях: от веба до разработки игр, от мобильного ПО до программ для корпораций и научных институтов.

1.4. Популярность Java

Вы уже знаете, что по результатам рейтинга State of Octoverse 2019 Java входит в тройку самых популярых языков программирования. «Джава» опережает по популярности PHP, C#, C++, TypeScript и другие востребованные языки, а уступает только JavaScript и Python.



Самые популярные языки программирования по данным State of Octoverse 2019

Справка: рейтинг State of Octoverse рассчитывается по количеству репозиториев на соответствующем языке, которые хранятся на GitHub.

По данным индекса TIOBE на ноябрь 2019 года, Java — самый популярный язык программирования. TIOBE — индикатор популярности языков программирования, который рассчитывается по сложной методике с учётом количества поисковых запросов, относящихся к тому или иному язку.

В рейтинге RedMonk Java занимает второе место, уступая JavaScript и опережая Python. Этот рейтинг рассчитывается на основе количества репозиториев на GitHub, как и State of Octoverse, однако методика ранжирования здесь отличается. Например, RedMonk не учитывает в расчётах форки репозиториев.

Такми образом, Java входит в тройку самых популярных языков программирования. По данным разных рейтингов, этот язык занимает места от первого до третьего.

Глава 2.Среды для разработки приложений в языке Java

2.1.. Некоторые важные термины Java SE, JRE, JDK,IDE

Говоря о языке программирования Java можно часто увидеть такие аббревиатуры как: Java SE, JRE, JDK, IDE. Рассмотрим их.

Java SE

Java Standard Edition (Java SE) – это стандартное издание Java, именно под него разрабатываются клиентские приложения. Приложения могут работать самостоятельно или в качестве апплетов веб-браузера.

Кроме стандартного издания бывают:

- Java Enterprise Edition (Java EE) для разработки приложений на стороне сервера, таких как Java сервлеты, JavaServer Pages (JSP) и JavaServer Faces (JSF).
- Java Micro Edition (Java ME) для разработки приложений под мобильные устройства, такие как телефоны.

JRE

Java Runtime Environment – среда выполнения Java. Это виртуальная машина Java, необходимая для запуска Java программ на компьютерах пользователя. В ней имеется всё, что необходимо для запуска Java приложений на вашей системе. JRE охватывает нужды большинства пользователей.

JDK

JDK (Java SE Development Kit) – набор инструментов разработчика для создания программ на Java. Включает в себя JRE плюс инструменты для разработки, отладки и мониторинга Java приложений.

Итак, чтобы определиться, что скачивать: JRE или JDK достаточно ответить на вопрос: нужно только запускать Java программы или ещё и разрабатывать их? Если только запускать, то достаточно JRE. Если вы разрабатывать программы, то необходим JDK. Набор разработчика JDK уже включает JRE, поэтому нет необходимости скачивать их оба по отдельности.

IDE

IDE (integrated development environment) – интегрированная среда разработки, предназначена для содействия разработчику, ускоряет процесс разработки программ. Обычно включает в себя редактор с подсветкой синтаксиса, справкой, автоматизированной компиляцией и запуском программ. IDE может быть создана для любого языка программирования, либо для конкретного языка, например, для Java. Интегрированная среды разработки бывают бесплатными и платными.

2.2. Среда разработки IntelliJ IDEA.

В Яве нет какой-то одной программы — среды разработки, поэтому в Сети встречаются примеры из разных программ. Самые примитивные, вроде Notepad++ , больше подходят для консольного вывода, но если рассматривать что-то более серьёзное, то выделяются только три кандидата: NetBeans, Eclipse и IntelliJ IDEA.

NetBeans — самая простая программа, которая быстро (относительно остальных) запускается и довольно сносно работает.

Eclipse —вариант, более мощный чем NetBeans, но слабее чем IntelliJ IDEA.

IntelliJ IDEA — выглядит самой крутой, но за это приходится платить скоростью работы. Стоит отметить, что Android Studio базируется на IntelliJ IDEA, но студия работает почему-то значительно медленней.

Важный момент связан с программированием под Андроид. Из этих трех IDE только IntelliJ IDEA для этого более-менее годится. В Сети масса материалов о программировании под Андроид в Eclipse, но они все уже устаревшие. Может быть старый Android SDK и будет работать, но все новые — уже нет.

В своей работе я решил использовать **IntelliJ IDEA.**

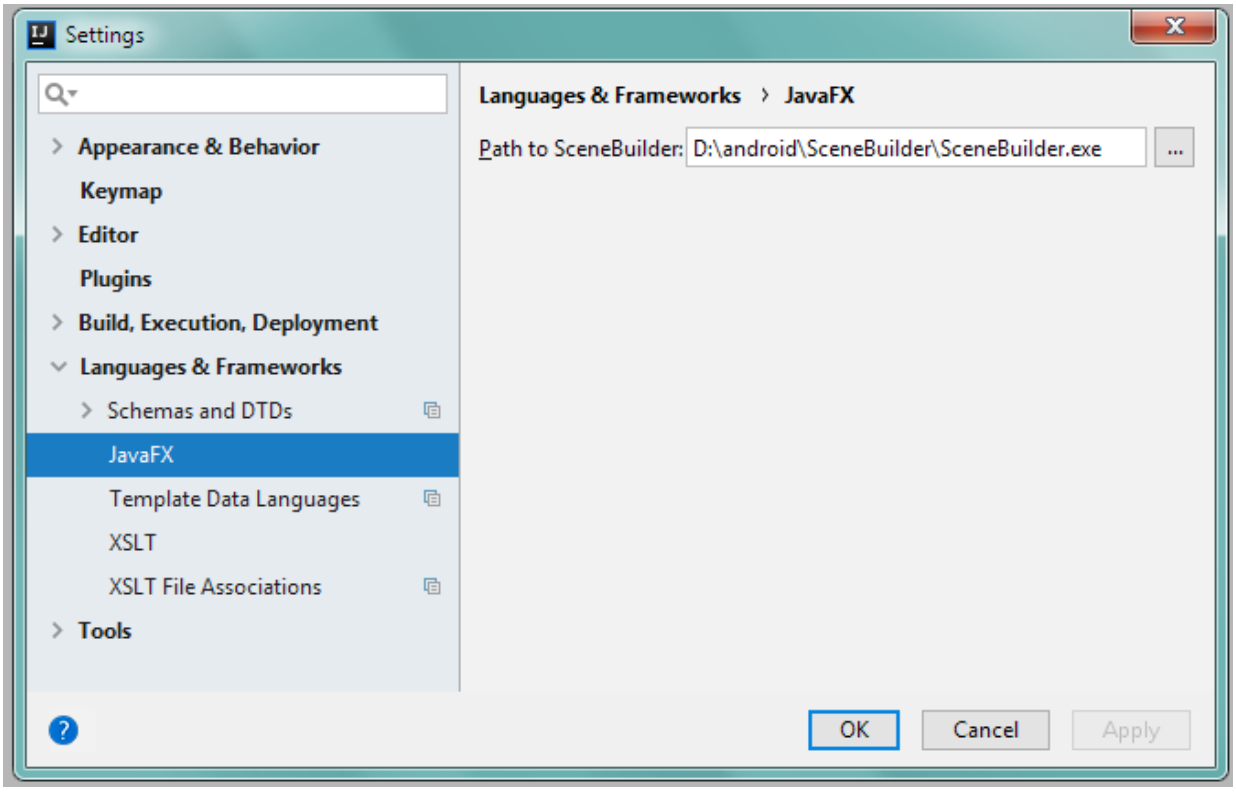
Рассмотрим, на простейшем примере, как работает IntelliJ IDEA. Создаваемая программа будет очень простой — кнопка и текстовое поле. При нажатии на кнопку в текстовое поле пусть добавляется текст «Привет!».

В программе нужно заранее прописать настройку — используемую JDK.

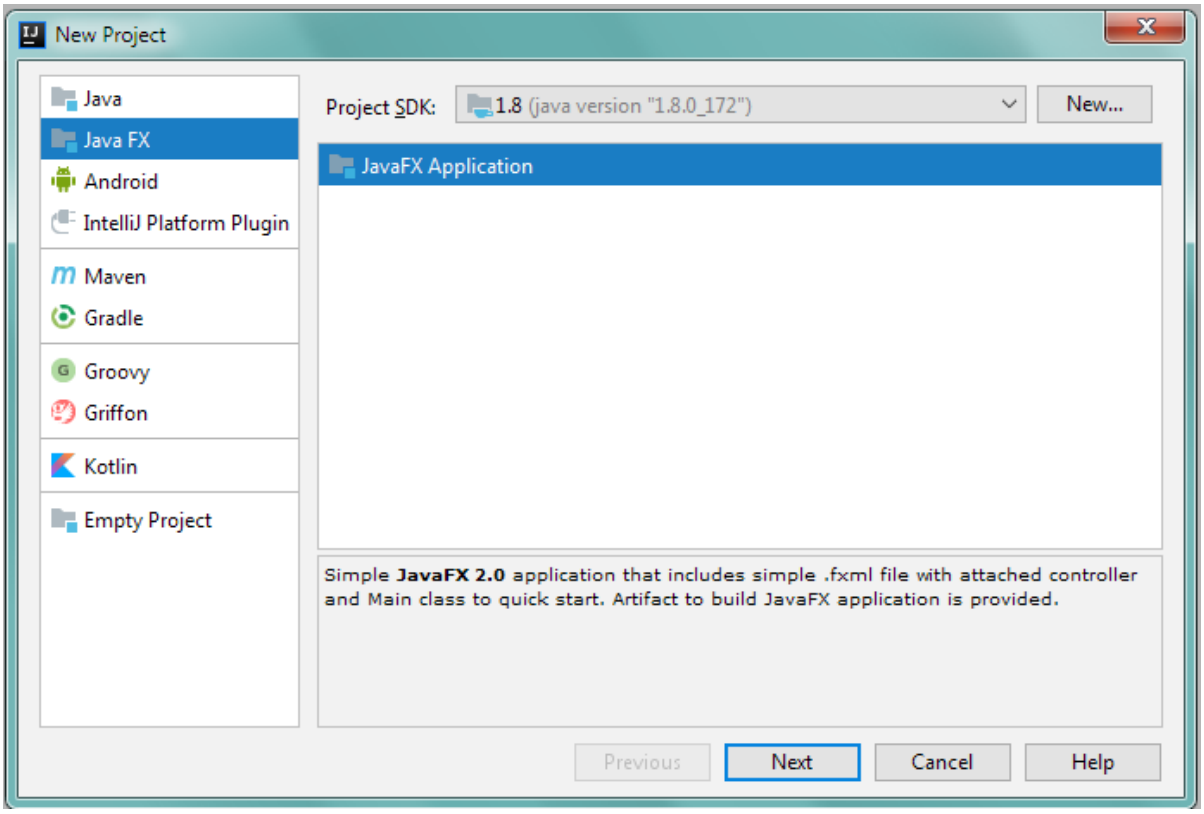
Н о прежде чем приступать, нужно установить ещё одну программу: SceneBuilder (используйте версию для Java 8). Это ключевая программа, где, собственно и выполняется всё визуальное построение. **SceneBuilder** самостоятельно может работать без IDE, поэтому можете её запустить и посмотреть как она устроена. На выходе будет fxml-файл, который содержит всю нужную разметку. Этот файл и используется в IDE, вместо написания кода.

IntelliJ IDEA запускается довольно неспешно.

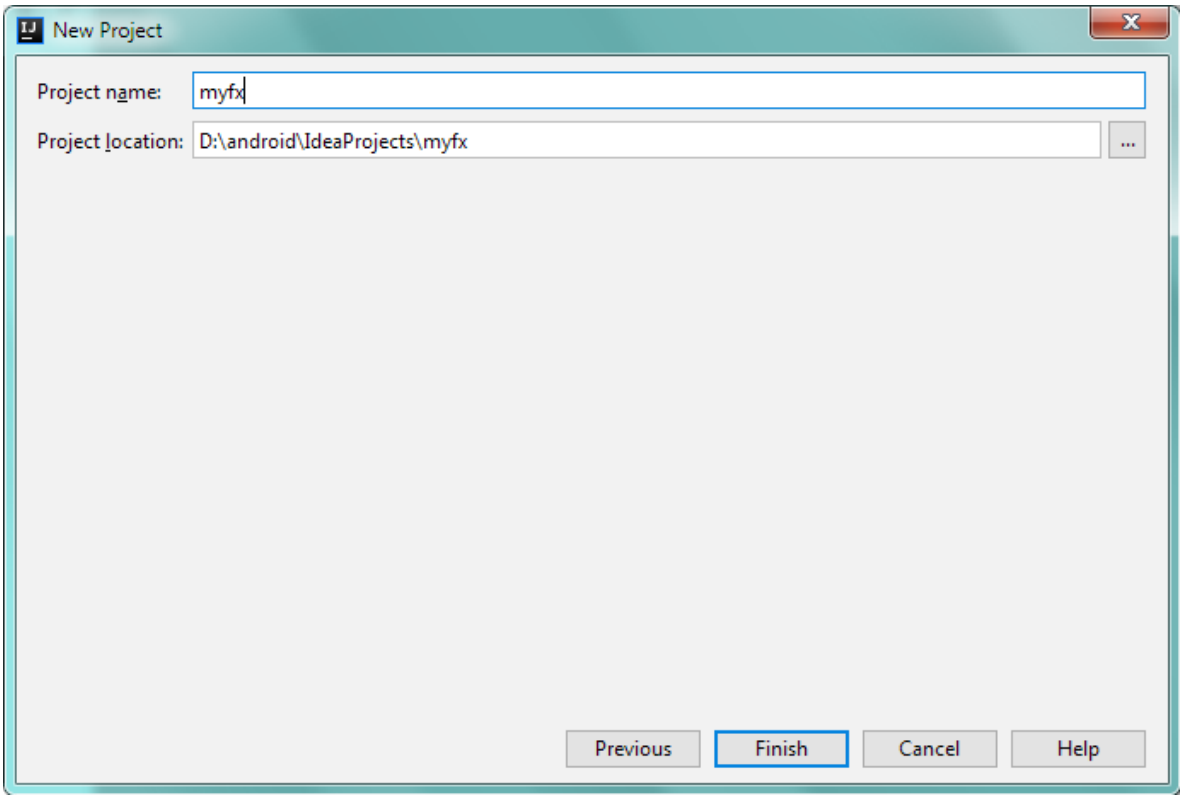
Указываем путь к SceneBuilder: *File - Settings*:



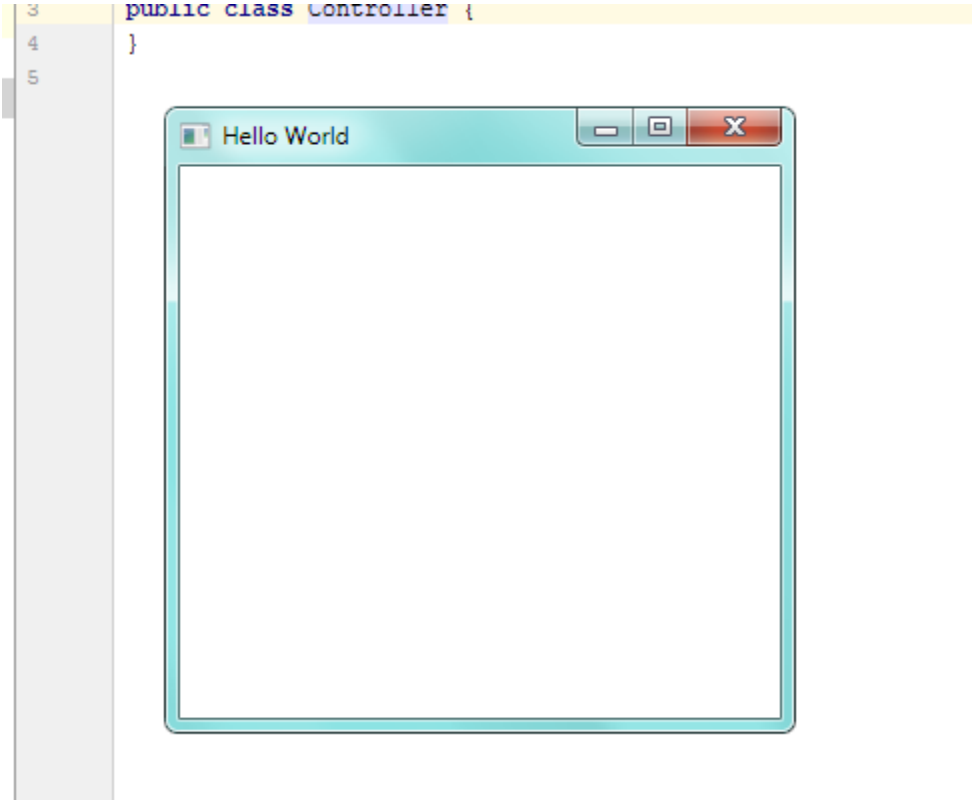
Дальше создаем новый проект:



И указываем его имя (как обычно — myfx):



IntelliJ IDEA выполнит инициализацию и мы увидим уже знакомые три файла. Запустим программу, чтобы проверить отсутствие ошибок.



Здесь также пустая форма, но зато указан заголовок программы. Если посмотреть на код *Main.java*, то увидим строчку:

```
primaryStage.setTitle("Hello World");
```

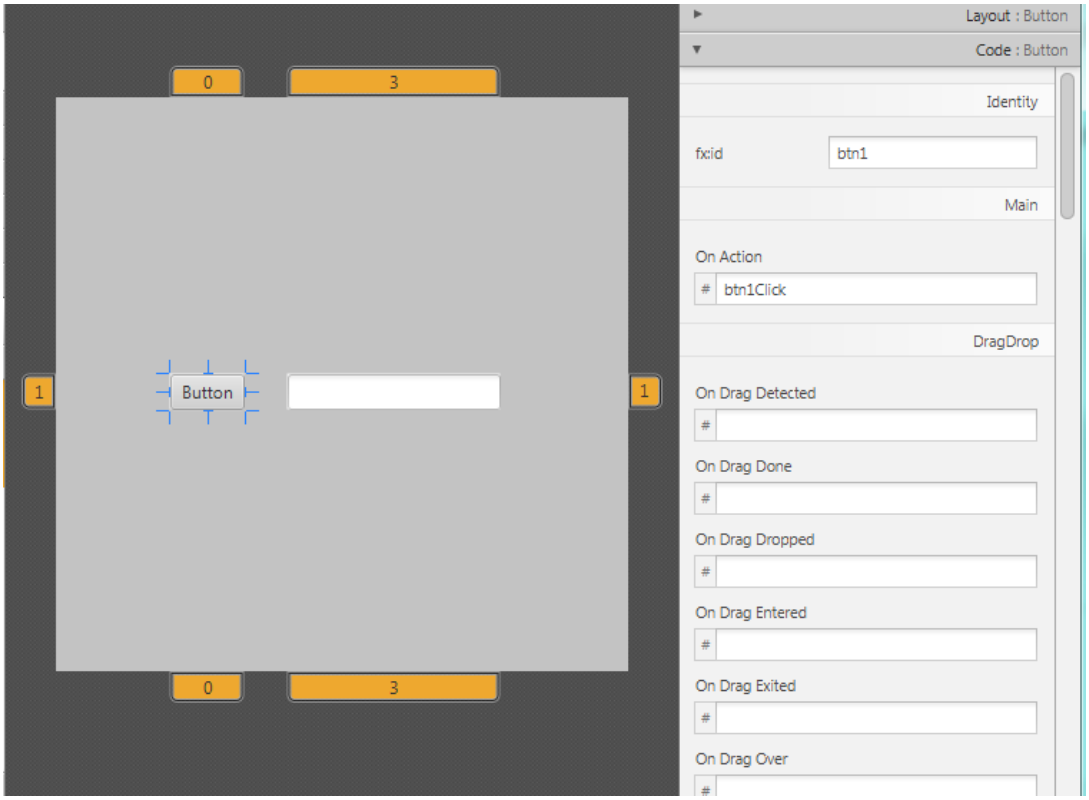
Это и есть заголовок приложения. В других IDE эта строчка отсутствует, но зато теперь мы знаем для чего нужны «театральные подмостки».

Переключаемся в SceneBuilder (аналогично Eclipse): второй кнопкой мыши нужно выбрать **Open in SceneBuilder**.

Здесь также пустая форма, но с контейнером **GridPane**. Устанавливаем **Pref Width** и **Pref Height**, чтобы отобразилась форма.

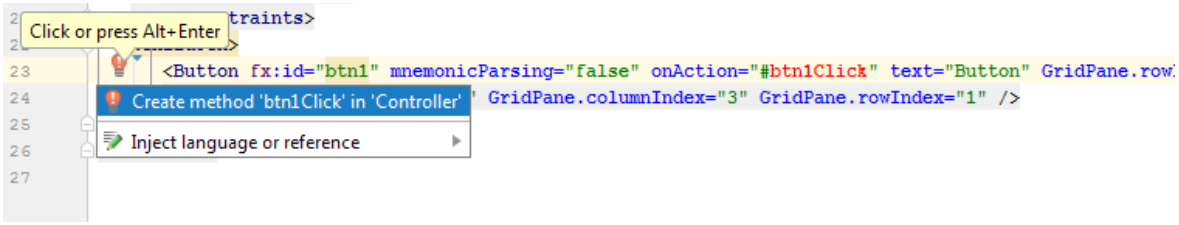
Сам по себе GridPane представляет собой аля-сетку из ячеек для элементов. Думаю, что тут нет смысла повторяться — все действия будут аналогичными:

- разместить кнопку и тестовое поле,
- присвоить им id,
- для кнопки прописать метод для обработки клика,
- не забываем проверить, указан ли контролёр (*sample.Controller*).

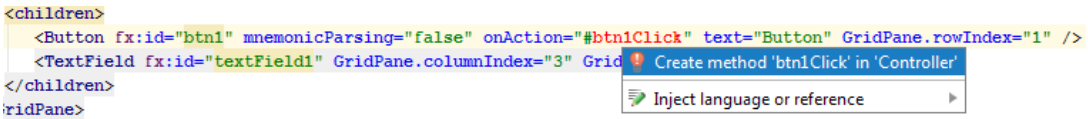


Закрываем SceneBuilder и возвращаемся в IntelliJ IDEA. Здесь нужно добавить идентификаторы id в код, а также создать метод для реакции на нажатие кнопки.

IntelliJ IDEA предлагает для этого два способа. Первый — при наведении мышки на «проблемное место» будет появляться подсказка, на которую можно кликнуть:

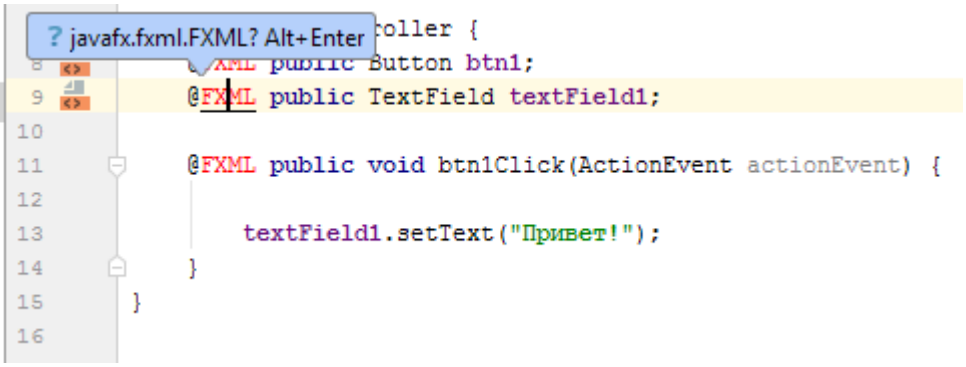


Второй — горячая клавиша **Alt+Enter**



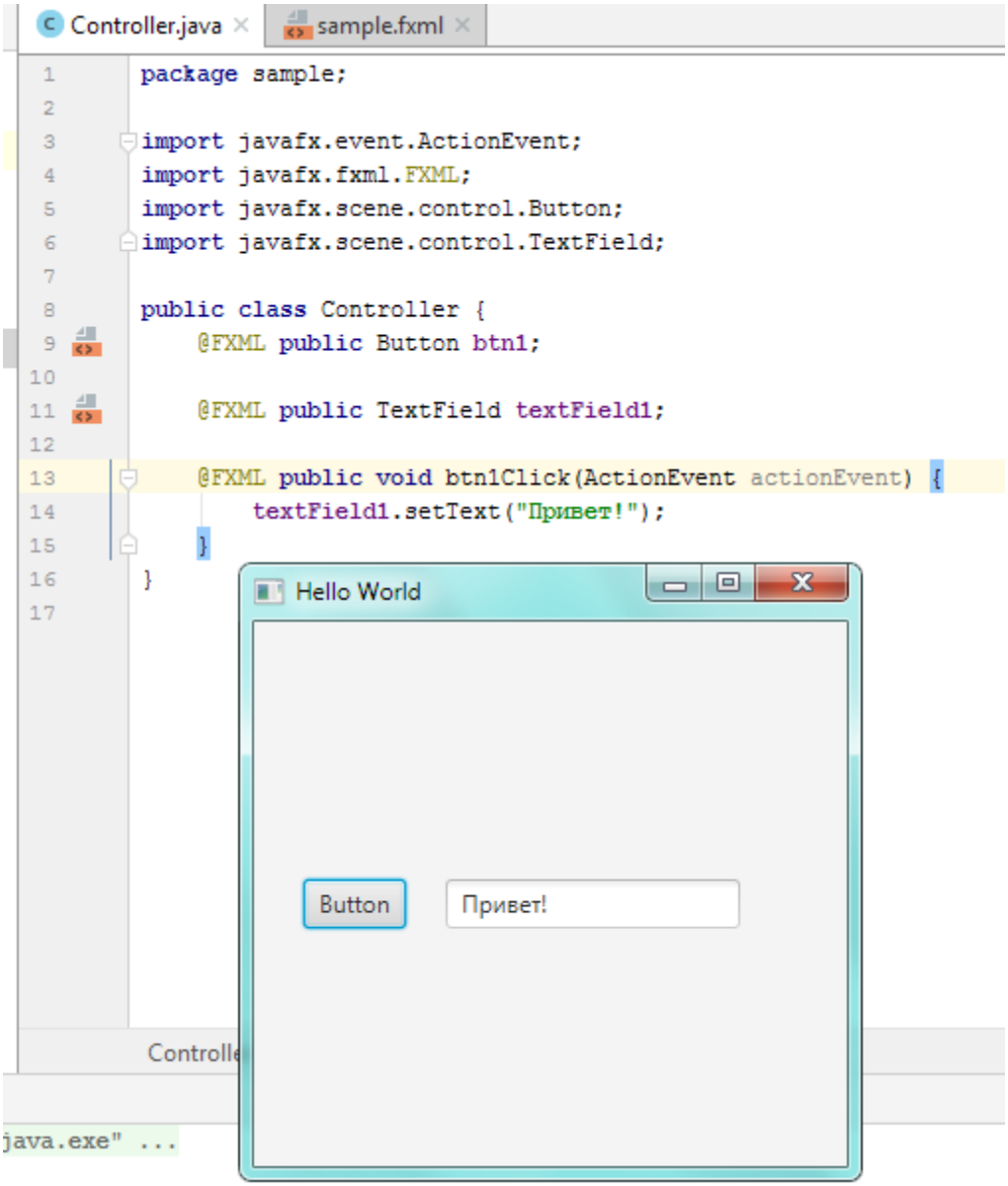
При каждом добавлении происходит переключение на файл контролёра, где можно сразу увидеть изменения.

При этом, заметьте, не добавляется строчка «@FXML». Если мы вручную его добавим перед переменными и методом, то IntelliJ IDEA сразу предложит добавить нужный java-класс:



Вообще IntelliJ IDEA показывает достаточно хорошую сообразительность и выступает помощником в написании кода. Причём делает это самостоятельно без лишних нажатий кнопок.

Добавляем свой код для обработчика и запускаем программу:



Простейшее окно готово. Теперь перейдем к разработке более серьезного приложения

Глава 3.Создание приложения в IntelliJ IDEA

Прежде чем приступить к созданию более сложного приложения, необходимо описать приблизительную структурную модель того, как всё будет работать. Отметим несколько пунктов, от которых будем в дальнейшем отталкиваться:

- Приложение будет представлять из себя несколько связанных между собой окон. Первым будет - главное меню, вторым – викторина, третьим и последующими – экскурсия по городу.
- Фактически, каждое из окон будет иметь различный дизайн, поэтому переход между ними будет осуществляться путём использования разных FXML файлов.
- Викторина будет состоять из последовательно заданных нами вопросов с подсветкой правильности ответа и из примечаний к этим вопросам после ответа пользователя.

Итак, приступим.

3.1 Главное окно

Сначала, по принципу описанному ранее, создаём проект, даём ему имя, а так же при необходимости указываем название контролирующего класса для главного окна в том случае, если изначально IntelliJ IDEA не сделала это автоматически. Вот строка с исходными параметрами в созданном нами *sample.fxml*:

```
ns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controller">
```

После этого переходим в SceneBuilder, и там открываем наш *sample.fxml*. В первую очередь удаляем элемент GridPane, что бы он не мешал в дальнейшем, а проблему центровки элементов решит сам JavaFX SceneBuilder, так как при изменении положения элемента в пространстве относительно других появляются направляющие, и программа автоматически выравнивает положение элемента.

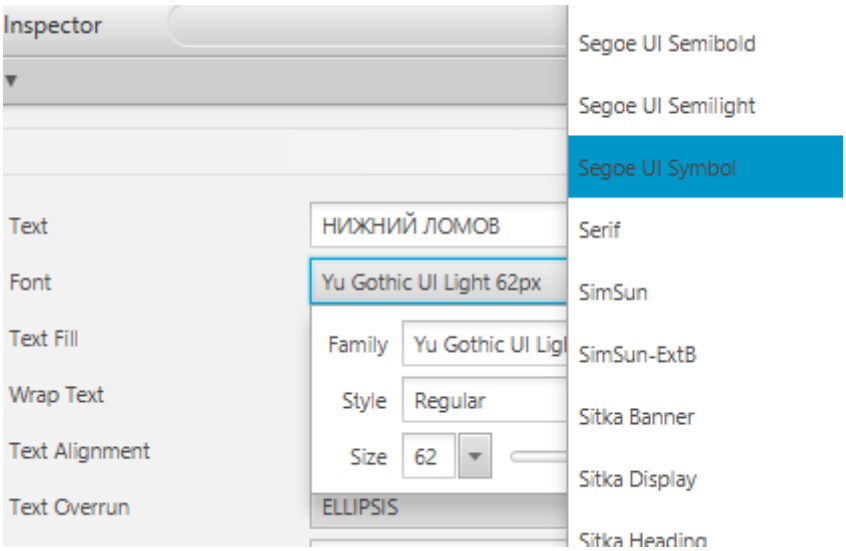
Размещаем элемент AnchorPane, который представляет из себя якорную панель, что позволяет привязывать элементы непосредственно к краям панели или в любой другой позиции. И сразу выставляем нужный нам размер путём изменения параметров Pref Width и Pref Hight в категории Layout. Выставить можно совершенно различные значения, которые могли бы соответствовать размерам монитора, я предпочёл окно 600x400 пикселей.

Далее выбираем или же создаём фон для главного окна. В качестве фона я выбрал вот эту картинку:

После того, как определились с фоном, делаем его форматом PNG и помещаем в папку *src* нашего проекта. Теперь переходим в SceneBuilder и размещаем элемент ImageView на нашей AnchorPane. В категории Properties нажимаем на три точки возле параметра Image и указываем путь до нашей картинки с фоном.

После этих манипуляций с общим видом окна размещаем на нём ещё 4 элемента: Label и 3 кнопки Button.

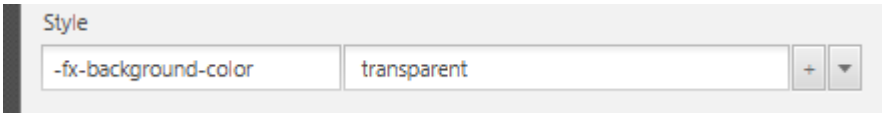
В Label указываем нашу надпись «НИЖНИЙ ЛОМОВ», изменяем цвет текста на белый, а шрифт выбираем из выпадающего списка:



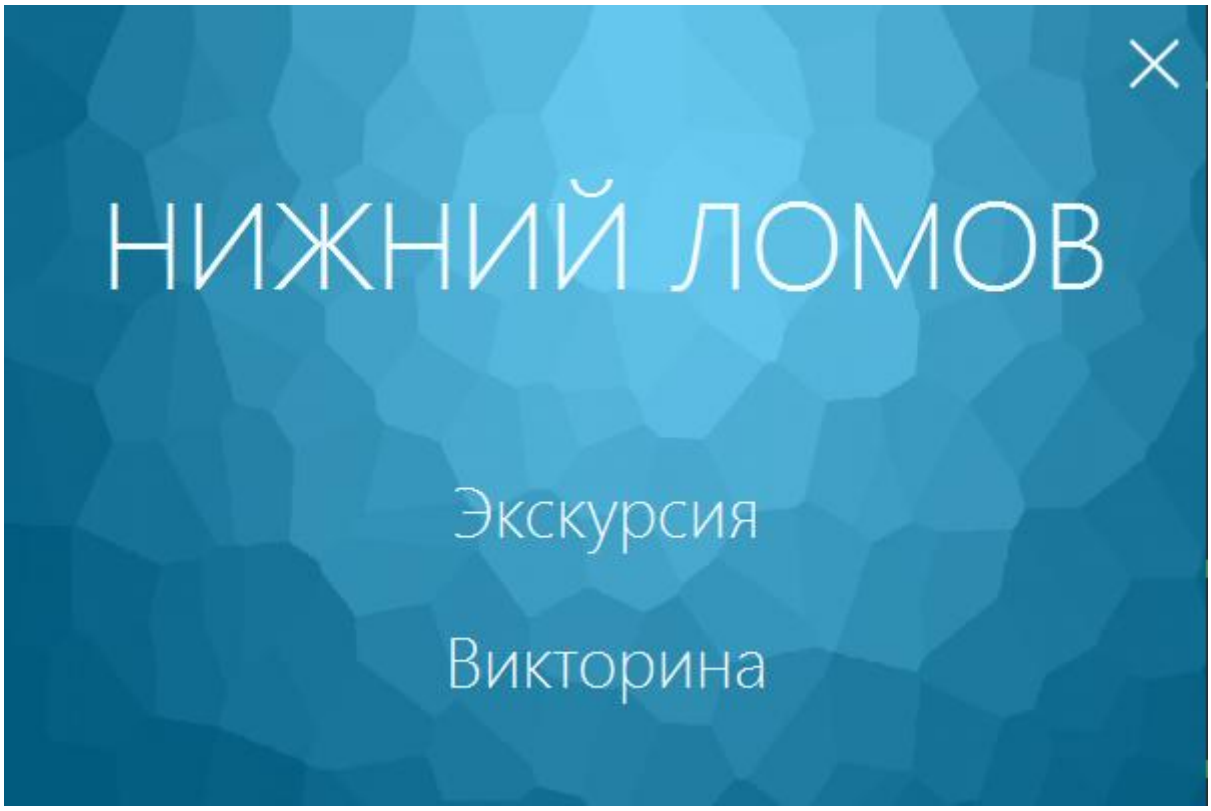
Размер текста устанавливаем по желанию.

Далее размещаем две кнопки по центру, а третью располагаем в правом верхнем углу. Две кнопки будут переводить нас на викторину и экскурсию, а

imageview2 будет закрывать программу. Для кнопок проводим аналогичные действия с текстом и даём им названия «Викторина» и «Экскурсия» соответственно, так же помимо этого, для более красивого дизайна находим в категории Properties параметр Style и вписываем туда следующие значения:



Данное действие позволит сделать фон кнопок прозрачным, что будет гораздо выигрышней по сравнению со стандартным стилем кнопок Java. Для imageview2 находим картинку белого крестика и по принципу, описанному ранее таким образом создаём почти «кнопку» выхода из программы. К кнопкам и «крестику» привязываем id как было сказано ранее. Вот что получилось в конечном итоге:



И так, дизайн создан, теперь переходим в IntelliJ IDEA для написания функционала этого окна. В классе Main сначала изменим наши основные параметры, дав главному окну надпись, отключив стандартное оформление окна в Windows, а также запретив изменение размеров окна:

```
Parent root = FXMLLoader.load(getClass().getResource( name: "sample.fxml"));
primaryStage.initStyle(StageStyle.UNDECORATED);
Scene scene = new Scene(root, width: 600, height: 400);
primaryStage.setTitle("nLomov");
primaryStage.setScene(scene);
primaryStage.show();
primaryStage.setResizable(false);
```

Далее в теле метода *initialize* по названному id ImageView2, который является у нас кнопкой закрытия, пишем лямбда-выражение *event -> {...},* которое будет закрывать приложение по нажатию на наш ImageView2.

```
void initialize() {
    wx.setOnMouseClicked(event -> {
        System.exit( status: 0);
    });
```

Теперь, рядом с файлом *sample.fxml* создаём ещё два файла формата fxml, назовём их *Quiz* и *sample2*. Эти файлы буду отвечать за окно викторины и окно экскурсии. Также необходимо создать контролирующие классы quiz.java и sample2.java соответственно, не стоит забывать про привязывание к контролирующим файлам наших fxml. После создания необходимых нам файлов, мы можем привязать их открытие по нажатию соответствующих кнопок аналогично при помощи лямбда выражения. Задаём размер и формат будущих

```
button1.setOnAction(event -> {

    button1.getScene().getWindow().hide();
    FXMLLoader loader = new FXMLLoader();
    loader.setLocation(getClass().getResource( name: "/sample/sample2.fxml"));
    try {
        loader.load();

    }
    catch (IOException e){
        e.printStackTrace();
    }
    Parent root = loader.getRoot();
    st.setScene(new Scene(root, width: 805, height: 745));
    st.setResizable(false);
    st.setTitle("Экскурсия по городу");
    st.show();
});
    button2.setOnAction(event2 -> {

        button2.getScene().getWindow().hide();
        FXMLLoader loader2 = new FXMLLoader();
        loader2.setLocation(getClass().getResource( name: "/sample/quiz.fxml"));
        try {
            loader2.load();

        }
        catch (IOException e){
            e.printStackTrace();
        }
        Parent root2 = loader2.getRoot();
        Stage st2 = new Stage();
        st2.setTitle("Тест на знание города");
        st2.setScene(new Scene(root2, width: 611, height: 461));
        st2.setResizable(false);
        st2.show();
});
```

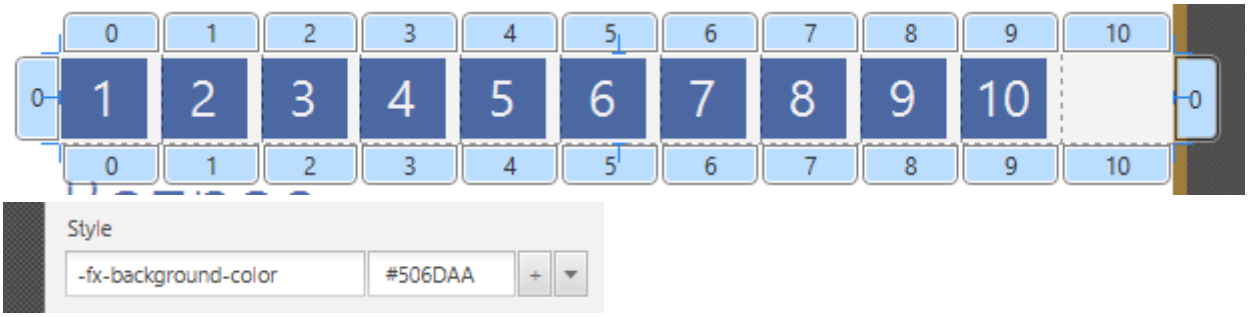
окон с помощью этого блока кода:

Теперь наше главное окно готово, переходим к созданию викторины.

3.2. Викторина

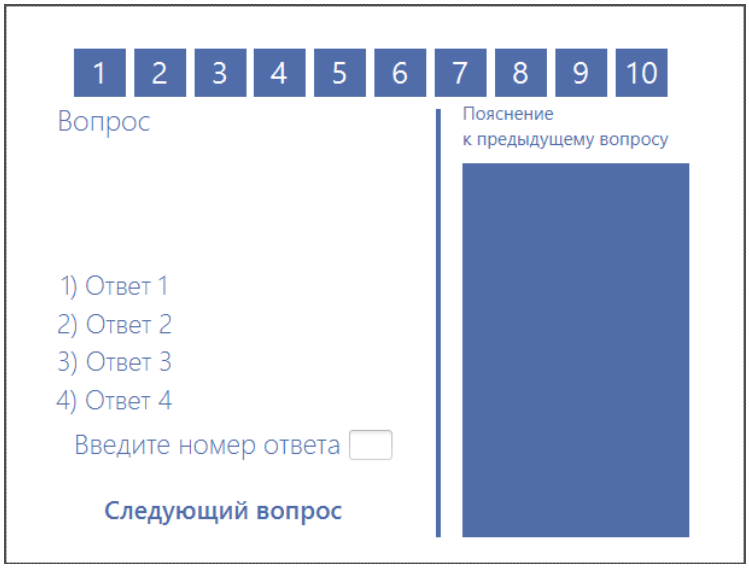
Создание окна викторины будет во многом схоже с созданием главного меню, за исключением того, что большую часть времени придётся уделить работе в IntelliJ IDEA. Итак, приступим.

Для начала откроем созданный нами ранее файл *Quiz.fxml* в SceneBuilder, аналогично создаём AnchorPane, но при это не удаляем GridPane, он нам ещё понадобится. Далее размещаем на AnchorPane следующие элементы: одна кнопка Button, текстовое поле TextField для ввода выбранного варианта номера ответа, 1 Label для вопроса, 4 Label для ответов и 2 Label для примечания к предыдущим вопросам, т.е. в качестве ответов для вопросов. Выставляем нужные нам шрифт и цвет, а также называем для того, чтобы в дальнейшем не путать элементы. Теперь переходим к GridPane и размещаем на ней 10 элементов Label, вписываем на них нумерацию, а также с помощью специального параметра background color задаём цвет фона цифрам и окошку под примечание:

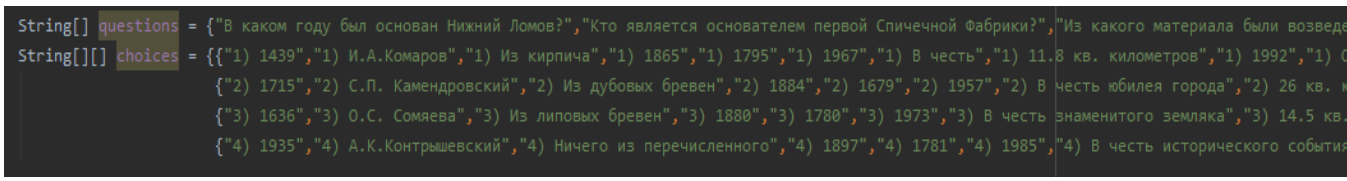


У элементов Label отвечающих за вопрос и примечание отмечаем галочку на Wrap Text в категории Properties, для того чтобы наш текст переносился на следующую строку в случае нехватки места.

Даём каждому Label и кнопке id и после этого в целом должно получиться вот это:



Для красоты я добавил дополнительно AnchorPane в качестве огораживающей полоски и дал ему цвет с помощью того самого background color, но делать это необязательно. Далее открываем наш IntelliJ IDEA и приступаем к написанию нашей викторины. Сначала создаём два строковых массива, один двумерный и один одномерный. В одномерном массиве будут хранится наши вопросы, а в двумерном соответственно ответы. Строка в двумерном массиве будет являться номерном ответа, а столбец - номером вопроса. Вот как это выглядит:



Помимо этого, создадим массив для численных переменных, и впишем туда по порядку номера ответов на наши вопросы. Ещё предварительно создадим две переменных и приравняем их к нулю. Это будут переменные score и i, пока они нам не понадобятся, но будут необходимы в дальнейшем. Так как вопросы будут идти последовательно, то каждое нажатие кнопки будет выполнять определённые действия, а в нашем случае менять вопрос на следующий, приписывать примечание к предыдущему вопросу, а также, в зависимости от ответа, менять цвет одной из верхних **Label**. С помощью лямбда выражения ***event ->{…}*** мы делаем 10 последовательных блоков, которые будут работать следующим образом:

1. Считывается ответ с текстового поля.

2. Считываемый ответ сравнивается с правильным по номеру массива через переменную ***i***.

3. Через условие if меняется цвет **Label** с номером вопроса.

4. Вопрос меняется на следующий по номеру массива через переменную ***i***.

5. Варианты ответов меняются на следующие по номеру массива через переменную ***i***.

6. Указывается примечание к предыдущему вопросу.

7. Переменная ***i*** увеличивается на 1.

8. Очищается текстовое поле **TextField**.

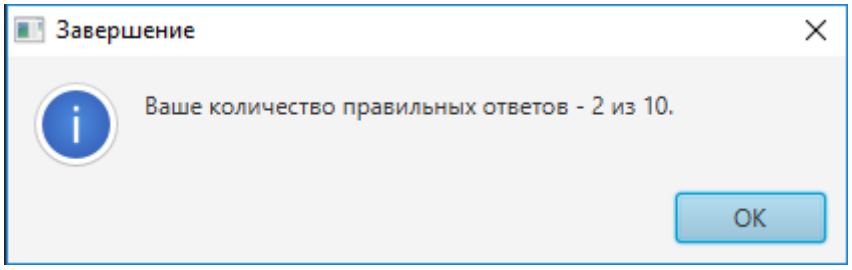
Вот пример готового блока кода:

```
question.setText(questions[i]);
a1.setText(choices[0][i]);
a2.setText(choices[1][i]);
a3.setText(choices[2][i]);
a4.setText(choices[3][i]);
slv.setOnAction(event1 -> {
    a = Integer.parseInt(ans.getText());
    if (a==answers[i]){
        score++;
        l2.setStyle("-fx-background-color: #08A400");}
    else
        l2.setStyle("-fx-background-color: #C61611");
    ans.clear();
    i++;
    p.setText("В 1858 крестьянин С. П. Камендровский основал спичечную фабрику, которая к концу века превратилась в крупное предприятие.");
```

После того, как пользователь закончит прохождение викторины, мне показалось нужным выводить количество правильных ответов в виде отдельной информации. Поэтому на всех блоках добавляем в условие строку ***score++;*** в том случае, если пользователь правильно ответит на вопрос. Далее создаем метод alerts с диалоговым окном, которое будет показывать результат. Вот как это может быть прописано в исходном коде:

```
private void alerts(int s) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Завершение");
    alert.setHeaderText(null);
    alert.setContentText("Ваше количество правильных ответов - "+s+" из 10.");
    alert.showAndWait();
}
```

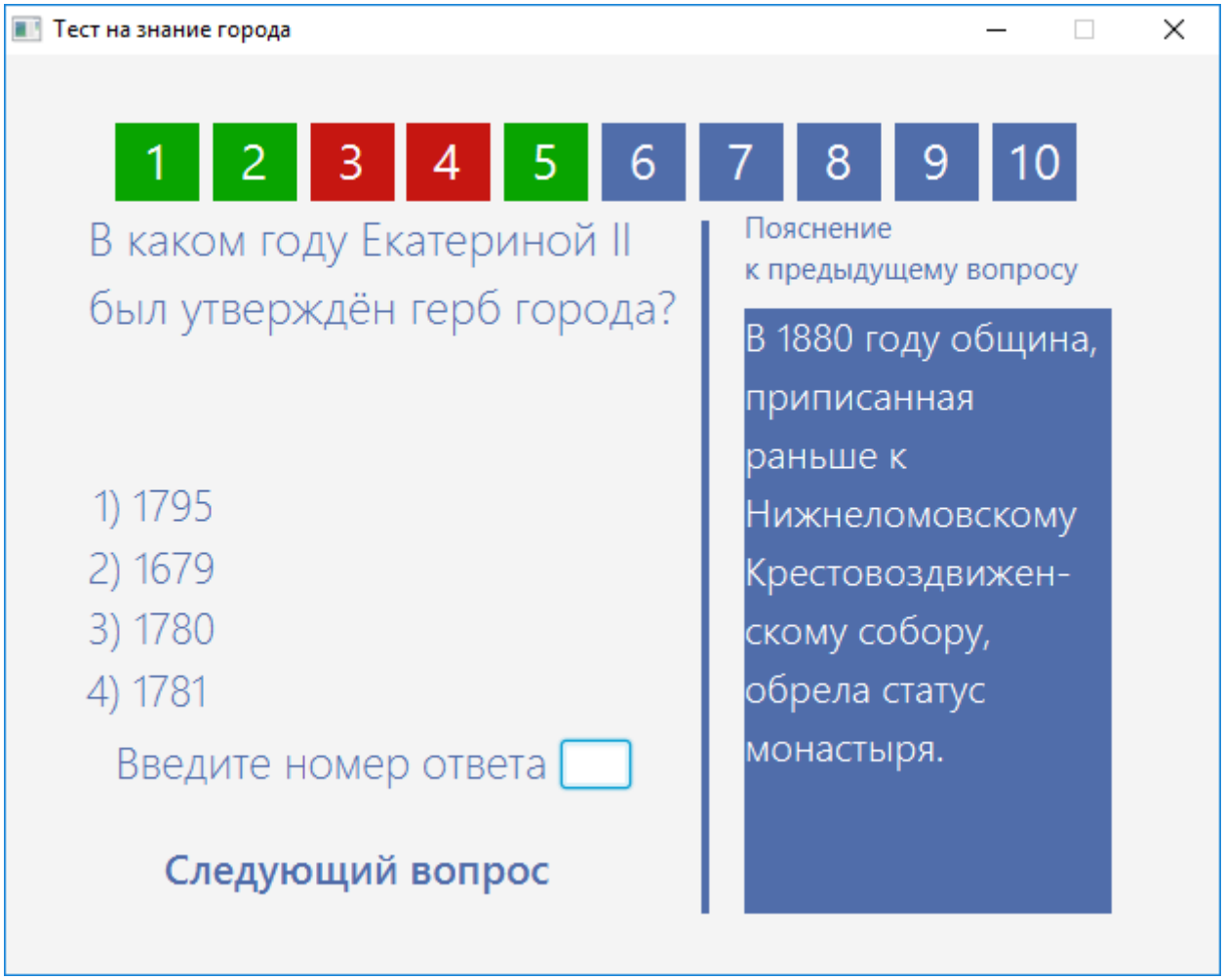
На вход метод принимает наш ***score***, который внутри обозначен как ***s***, а на выход выводит диалоговое окно с информацией:



Строку вызова метода напишем в конце, так же по нажатию кнопки на последний вопрос изменим текст кнопки на «Завершить», а потом на «Выход»:

```
slv.setText("Завершить");
slv.setOnAction(event10 -> {
    alerts(score);
    slv.setText("Выход");
    slv.setOnAction(event11 -> {
        System.exit( status: 0);
    });
```

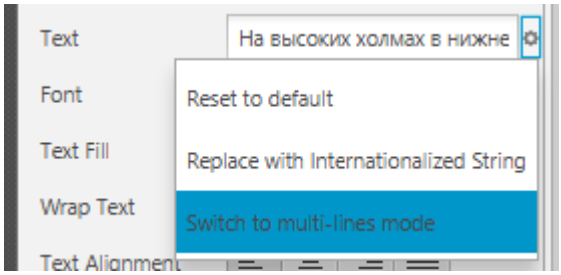
Тестируем, и получаем готовую викторину с 10 вопросами, с подсчётом и подсветом правильных ответов:



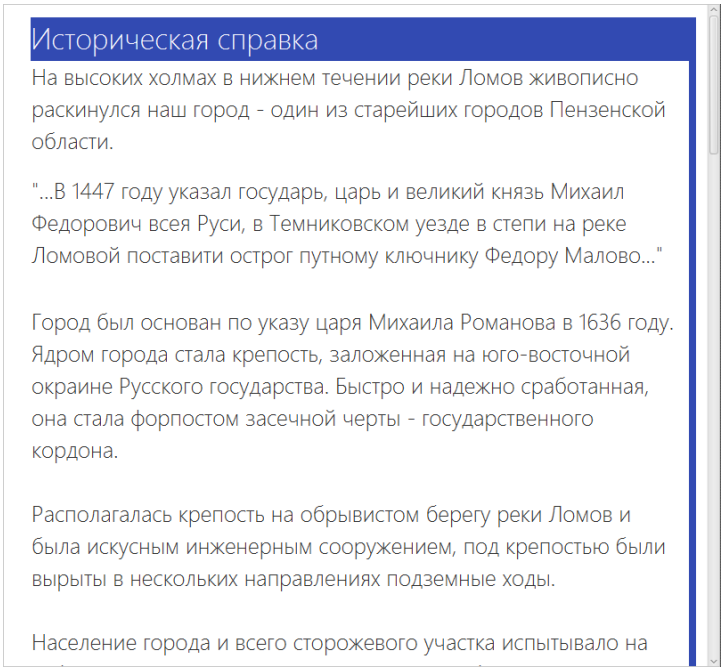
Следующим этапом в создании приложения будет создание окон с экскурсией по городу.

3.3. Экскурсия по городу

Теперь переходим к созданию экскурсии но Нижнему Ломову. Разделим её на части, первой из которых будет «Историческая справка», с неё и начнём. Аналогично предыдущим пунктам открываем SceneBuilder, удаляем **GridPane**, добавляем **AnchorPane**, и начинаем с того, что изменим размер нашего окна в **Layout** на 789x3370. Такой большой размер нужен нам чтобы уместить всю информацию, которую мы хотим использовать. Добавляем Button, перекрашиваем текст и задний фон кнопки и соответственно изменяем надпись на «Следующая страница». Как вы уже поняли, эта кнопка позволит нам открывать последующие страницы экскурсии. Далее создаём Label для заголовка, так же его перекрашиваем и добавляем **AnchorPane** в виде полосы на всё окно. Теперь просто поочерёдно добавляем Label и вписываем нужный нам текст, для удобства в параметрах текста выбираем *Switch to multi-line mode*:



Таким образом дописываем всю информацию, которую хотим видеть в экскурсии, добавляем картинки через **ImageView**, оформляем через **AnchorPane** и в итоге у нас получилась длинная полоса с исторической справкой. Далее находим элемент **ScrollPane** и наносим его поверх нашего главного **AnchorPane**, растягиваем, и у нас получается готовое окно с исторической справкой о городе:



Но при этом возникает странная ошибка Java, из-за которой наш **ScrollPane** при открытии прокручивается вниз, к сожалению, решения у этой проблемы нет, но это не столь критично и вполне поправимо путём добавления кнопки «Наверх» с изменением значения **ScrollPane**:

```
naverh.setOnAction(event -> {
    scrollpane.setVvalue(0.0);
});
```

Готово, не забываем дать id кнопкам и переходим в IntelliJ IDEA. Заранее создадим класс *sample3.java* и *sample3.fxml*, там будет располагаться следующее окно с экскурсией по историческим местам. Теперь обратимся к нашей кнопке, и в силу того, что мы находимся не в главном классе, а во второстепенном, то воспользоваться ***FXMLLoader*** как отдельным объектом мы не можем, поэтому обойдём данную проблему путём введения ***try {…} catch() {…}***, чтобы в случае ошибки моментально выявить нашу проблему. Создаём наш контейнер Parent и

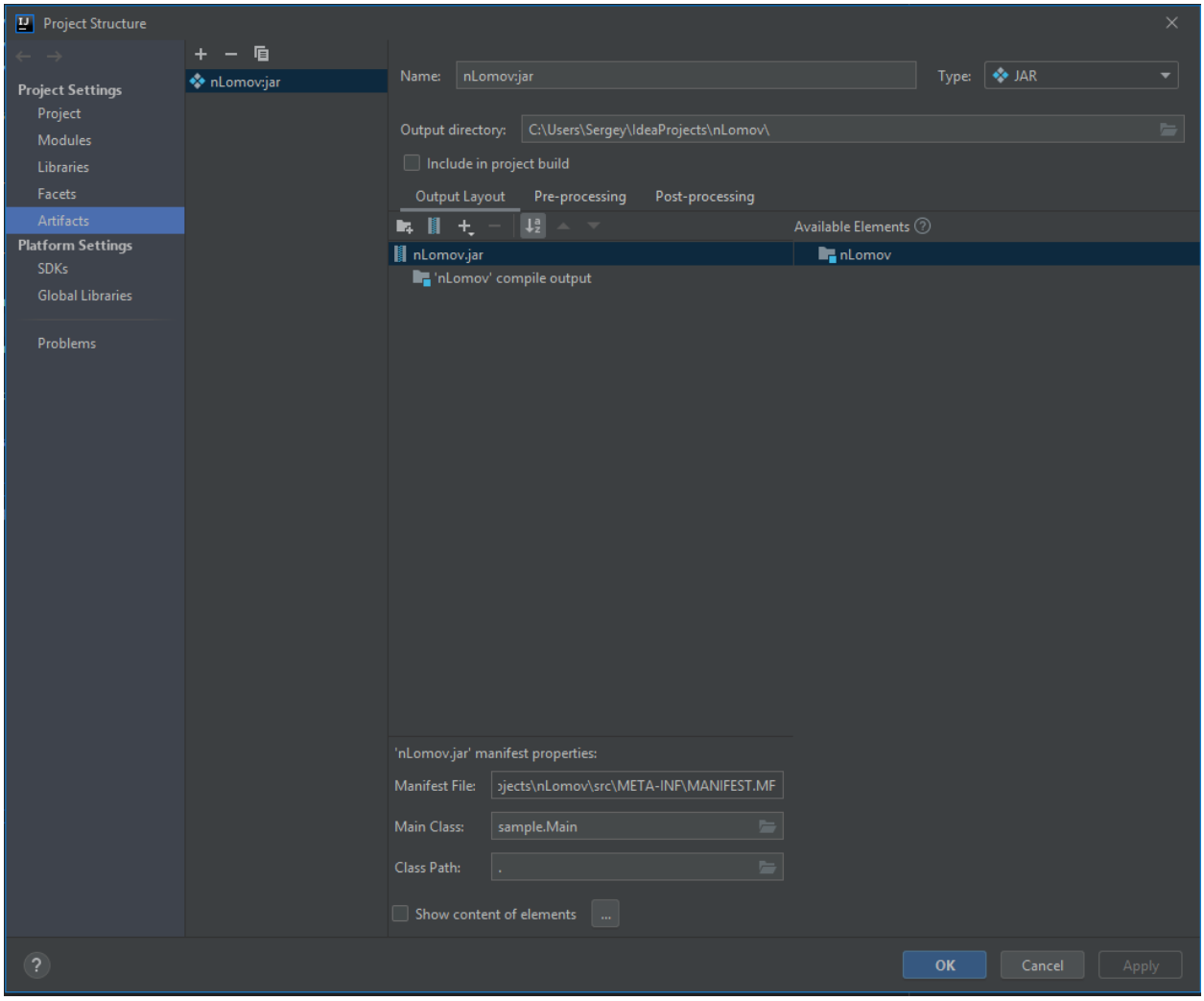
далее присваиваем ему загрузку нашего следующего *sample3.fxml* через ***FXMLLoader***, а всё остальное добавляем как обычно:

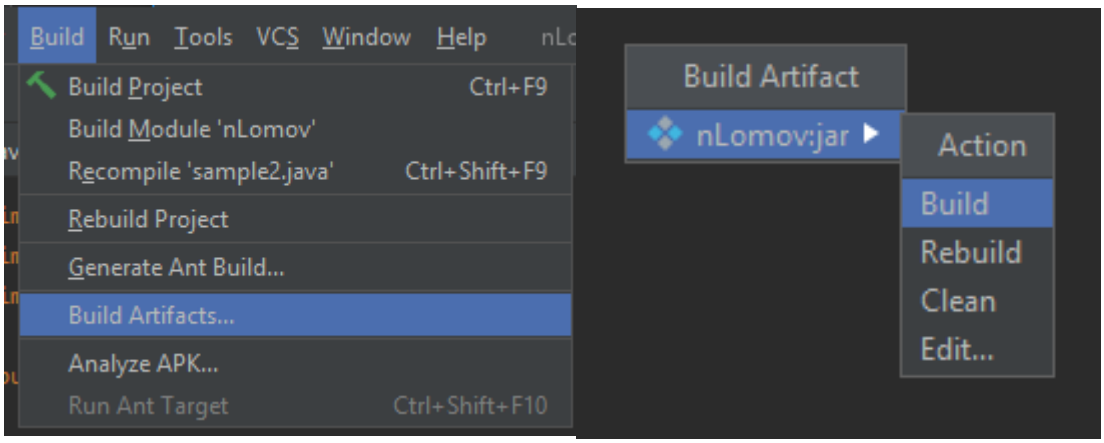
```
npage.setOnAction(event1 -> {

    Parent r3 = null;
    try {
        r3 = FXMLLoader.load(getClass().getResource( name: "/sample/sample3.fxml"));
    } catch (IOException e) {
        e.printStackTrace();
    }
    Stage okno2 = new Stage();
    Scene s2 = new Scene(r3, width: 805, height: 745);
    okno2.setScene(s2);
    okno2.setTitle("Экскурсия по достопримечательностям");
    okno2.showAndWait();
});
```

Аналогично оформляем следующее окно и проделываем те же действия. Теперь наше приложение готово, остался последний шаг – сформировать готовый файл для запуска.

3.4 Создание исполняемого JAR файла.

IntelliJ IDEA предлагает в качестве своих возможностей создание исполняемого файла, что сильно облегчит нам задачу. Сначала во вкладке **File** выбираем пункт **Project Structure**, далее в открывшемся окне находим пункт **Artifacts**. Нажимаем на плюс и из списка выбираем JAR, а потом *From modules with dependencies...* Далее в окне выбираем наш **Main** класс, он там будет один и жмём «ОК». После этого вверху указываем путь к файлу, где будет формироваться наш JAR архив и жмём «Apply». Теперь в верхнем меню находим Build и выбираем *Build Artifacts*. Из выпадающего списка выбираем Build ждём. Важно чтобы путь к файлу, где будет сформирован наш JAR архив имел только латинские символы.





Готово, проверяем, запускается ли приложение, всё ли работает и т.д. И на этом завершается создание приложения.

Заключение

Главный вывод — **на Jave возможно визуальное программирование**, особенно для новичков. Изучать язык гораздо интереснее, когда есть какой-то осязаемый результат — ООП, классы и прочие премудрости — это хорошо, но лучше начать с кнопочек, полей ввода, меню и всего того, что подразумевает программа.

Рассмотренная программа IntelliJ IDEA подходит для самостоятельного изучения, при условии, что имеются первоначальные навыки программирования.

Список используемых источников информации

1. 1https://code.makery.ch/ru/library/javafx-tutorial/part1/
2. https://java9.ru/?p=1
3. https://maxsite.org/page/visual-programming-java
4. https://ru.hexlet.io/blog/posts/yazyk-programmirovaniya-java-osobennosti-populyarnost-situatsiya-na-rynke-truda
5.