Бесплатная электронная книга

учусь

javafx

Free unaffiliated eBook created from
Stack Overflow contributors.

#javafx

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: javafx

It is an unofficial and free javafx ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official javafx.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с javafx

замечания

JavaFX - это программная платформа для создания и доставки настольных приложений, а также богатых интернет-приложений (RIA), которые могут работать на самых разных устройствах. JavaFX предназначен для замены Swing в качестве стандартной библиотеки GUI для Java SE.

ИТ позволяет разработчикам разрабатывать, создавать, тестировать, отлаживать и развертывать богатые клиентские приложения.

Внешний вид приложений JavaFX можно настроить с помощью каскадных таблиц стилей (CSS) для стилизации (см. JavaFX: CSS) и (F) XML-файлы можно использовать для создания объектов, упрощающих создание или разработку приложения (см. FXML и контроллеры) , Scene Builder - это визуальный редактор, позволяющий создавать файлы fxml для пользовательского интерфейса без написания кода.

Версии

Версия	Дата выхода
JavaFX 2	2011-10-10
JavaFX 8	2014-03-18

Examples

Установка или настройка

> API JavaFX доступны как полностью интегрированная функция Java SE Runtime Environment (JRE) и Java Development Kit (JDK). Поскольку JDK доступен для всех основных настольных платформ (Windows, Mac OS X и Linux), приложения JavaFX, скомпилированные для JDK 7 и более поздних версий, также работают на всех основных настольных платформах. Поддержка платформ ARM также была доступна с помощью JavaFX 8. JDK для ARM включает базовые, графические и управляющие компоненты JavaFX.

Чтобы установить JavaFX, установите выбранную вами версию среды Java Runtime и набора Java Development .

Возможности JavaFX включают:

1. Java API.
2. FXML и Scene Builder.
3. WebView.
4. Совместимость Swing.
5. Встроенные элементы пользовательского интерфейса и CSS.
6. Тема Модены.
7. Особенности 3D-графики.
8. API холста.
9. Печать API.
10. Поддержка Rich Text.
11. Поддержка Multitouch.
12. Поддержка Hi-DPI.
13. Графический конвейер с аппаратным ускорением.
14. Высокопроизводительный медиа-движок.
15. Автономная модель развертывания приложений.

Программа Hello World

Следующий код создает простой пользовательский интерфейс, содержащий одну `Button` которая печатает `String` на консоли при нажатии.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {

    @Override
    public void start(Stage primaryStage) {
        // create a button with specified text
        Button button = new Button("Say 'Hello World'");

        // set a handler that is executed when the user activates the button
        // e.g. by clicking it or pressing enter while it's focused
        button.setOnAction(e -> {
           //Open information dialog that says hello
           Alert alert = new Alert(AlertType.INFORMATION, "Hello World!?");
           alert.showAndWait();
        });

        // the root of the scene shown in the main window
        StackPane root = new StackPane();

        // add button as child of the root
        root.getChildren().add(button);

        // create a scene specifying the root and the size
        Scene scene = new Scene(root, 500, 300);
```

```
        // add scene to the stage
        primaryStage.setScene(scene);

        // make the stage visible
        primaryStage.show();
    }

    public static void main(String[] args) {
        // launch the HelloWorld application.

        // Since this method is a member of the HelloWorld class the first
        // parameter is not required
        Application.launch(HelloWorld.class, args);
    }

}
```

Класс `Application` является точкой входа для каждого приложения JavaFX. Можно запускать только одно `Application` , и это делается с использованием

```
Application.launch(HelloWorld.class, args);
```

Это создает экземпляр класса `Application` переданного как параметр, и запускает платформу JavaFX.

Для программиста важно следующее:

1. Первый `launch` создает новый экземпляр класса `Application` (`HelloWorld` в этом случае). Поэтому для класса `Application` нужен конструктор no-arg.
2. `init()` вызывается в созданном экземпляре `Application` . В этом случае реализация по умолчанию из `Application` ничего не делает.
3. `start` вызывается для экземпляра `Appication` и первичный `Stage` (= окно) передается методу. Этот метод автоматически вызывается в потоке приложения JavaFX (поток платформы).
4. Приложение запускается до тех пор, пока платформа не решит, что пришло время закрыть. Это делается, когда последнее окно закрывается в этом случае.
5. Метод `stop` вызывается в экземпляре `Application` . В этом случае реализация из `Application` ничего не делает. Этот метод автоматически вызывается в потоке приложения JavaFX (поток платформы).

В методе `start` построен граф сцены. В этом случае он содержит 2 `Button` `Node` : A и `StackPane` .

`Button` представляет собой кнопку в пользовательском интерфейсе, а `StackPane` - это контейнер для `Button` который определяет его размещение.

`Scene` создана для отображения этих `Node` . Наконец, `Scene` добавляется в `Stage` которая является окном, отображающим весь пользовательский интерфейс.

Прочитайте Начало работы с javafx онлайн: https://riptutorial.com/ru/javafx/topic/887/начало-работы-с-javafx

глава 2: CSS

Синтаксис

- Селектор NodeClass / * класса Node * /
- .someclass / * селектор по классу * /
- #someId / * селектор по id * /
- [selector1]> [селектор2] / * для прямого дочернего узла, соответствующего селектору 1, который соответствует selector2 * /
- [селектор1] [селектор2] / * для потомка узла, соответствующего селектору 1, который соответствует selector2 * /

Examples

Использование CSS для стилизации

CSS можно применять в нескольких местах:

- inline (`Node.setStyle`)
- в таблице стилей
  - К `Scene`
    - как таблица стилей пользовательского агента (не показана здесь)
    - как «нормальная» таблица стилей для `Scene`
  - К `Node`

Это позволяет изменять изменяемые свойства `Nodes` . Следующий пример демонстрирует это:

Класс приложения

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Region;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class StyledApplication extends Application {

    @Override
    public void start(Stage primaryStage) {

        Region region1 = new Region();
        Region region2 = new Region();
        Region region3 = new Region();
        Region region4 = new Region();
```

```
        Region region5 = new Region();
        Region region6 = new Region();

        // inline style
        region1.setStyle("-fx-background-color: yellow;");

        // set id for styling
        region2.setId("region2");

        // add class for styling
        region2.getStyleClass().add("round");
        region3.getStyleClass().add("round");

        HBox hBox = new HBox(region3, region4, region5);

        VBox vBox = new VBox(region1, hBox, region2, region6);

        Scene scene = new Scene(vBox, 500, 500);

        // add stylesheet for root
        scene.getStylesheets().add(getClass().getResource("style.css").toExternalForm());

        // add stylesheet for hBox
        hBox.getStylesheets().add(getClass().getResource("inlinestyle.css").toExternalForm());

        scene.setFill(Color.BLACK);

        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

inlinestyle.css

```
* {
    -fx-opacity: 0.5;
}

HBox {
    -fx-spacing: 10;
}

Region {
    -fx-background-color: white;
}
```

style.css

```
Region {
    width: 50;
    height: 70;

    -fx-min-width: width;
    -fx-max-width: width;
```

```
    -fx-min-height: height;
    -fx-max-height: height;

    -fx-background-color: red;
}

VBox {
    -fx-spacing: 30;
    -fx-padding: 20;
}

#region2 {
    -fx-background-color: blue;
}
```

Расширение Rectangle с добавлением новых стилируемых свойств

JavaFX 8

В следующем примере показано, как добавить пользовательские свойства, которые можно создать из CSS в пользовательский `Node` .

Здесь 2 `DoubleProperty` добавлены в класс `Rectangle` чтобы разрешить установку `width` и `height` из CSS.

Для стилизации пользовательского узла можно использовать следующий CSS:

```
 StyleableRectangle {
    -fx-fill: brown;
    -fx-width: 20;
    -fx-height: 25;
    -fx-cursor: hand;
}
```

Пользовательский узел

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import javafx.beans.property.DoubleProperty;
import javafx.css.CssMetaData;
import javafx.css.SimpleStyleableDoubleProperty;
import javafx.css.StyleConverter;
import javafx.css.Styleable;
import javafx.css.StyleableDoubleProperty;
import javafx.css.StyleableProperty;
import javafx.scene.paint.Paint;
import javafx.scene.shape.Rectangle;

public class StyleableRectangle extends Rectangle {

    // declaration of the new properties
    private final StyleableDoubleProperty styleableWidth = new
```

```
SimpleStyleableDoubleProperty(WIDTH_META_DATA, this, "styleableWidth");
    private final StyleableDoubleProperty styleableHeight = new
SimpleStyleableDoubleProperty(HEIGHT_META_DATA, this, "styleableHeight");

    public StyleableRectangle() {
        bind();
    }

    public StyleableRectangle(double width, double height) {
        super(width, height);
        initStyleableSize();
        bind();
    }

    public StyleableRectangle(double width, double height, Paint fill) {
        super(width, height, fill);
        initStyleableSize();
        bind();
    }

    public StyleableRectangle(double x, double y, double width, double height) {
        super(x, y, width, height);
        initStyleableSize();
        bind();
    }

    private void initStyleableSize() {
        styleableWidth.set(getWidth());
        styleableHeight.set(getHeight());
    }

    private final static List<CssMetaData<? extends Styleable, ?>> CLASS_CSS_META_DATA;

    // css metadata for the width property
    // specify property name as -fx-width and
    // use converter for numbers
    private final static CssMetaData<StyleableRectangle, Number> WIDTH_META_DATA = new
CssMetaData<StyleableRectangle, Number>("-fx-width", StyleConverter.getSizeConverter()) {

        @Override
        public boolean isSettable(StyleableRectangle styleable) {
            // property can be set iff the property is not bound
            return !styleable.styleableWidth.isBound();
        }

        @Override
        public StyleableProperty<Number> getStyleableProperty(StyleableRectangle styleable) {
            // extract the property from the styleable
            return styleable.styleableWidth;
        }
    };

    // css metadata for the height property
    // specify property name as -fx-height and
    // use converter for numbers
    private final static CssMetaData<StyleableRectangle, Number> HEIGHT_META_DATA = new
CssMetaData<StyleableRectangle, Number>("-fx-height", StyleConverter.getSizeConverter()) {

        @Override
        public boolean isSettable(StyleableRectangle styleable) {
            return !styleable.styleableHeight.isBound();
```

```
        }

        @Override
        public StyleableProperty<Number> getStyleableProperty(StyleableRectangle styleable) {
            return styleable.styleableHeight;
        }
    };

    static {
        // combine already available properties in Rectangle with new properties
        List<CssMetaData<? extends Styleable, ?>> parent = Rectangle.getClassCssMetaData();
        List<CssMetaData<? extends Styleable, ?>> additional = Arrays.asList(HEIGHT_META_DATA,
WIDTH_META_DATA);

        // create arraylist with suitable capacity
        List<CssMetaData<? extends Styleable, ?>> own = new ArrayList(parent.size()+
additional.size());

        // fill list with old and new metadata
        own.addAll(parent);
        own.addAll(additional);

        // make sure the metadata list is not modifiable
        CLASS_CSS_META_DATA = Collections.unmodifiableList(own);
    }

    // make metadata available for extending the class
    public static List<CssMetaData<? extends Styleable, ?>> getClassCssMetaData() {
        return CLASS_CSS_META_DATA;
    }

    // returns a list of the css metadata for the stylable properties of the Node
    @Override
    public List<CssMetaData<? extends Styleable, ?>> getCssMetaData() {
        return CLASS_CSS_META_DATA;
    }

    private void bind() {
        this.widthProperty().bind(this.styleableWidth);
        this.heightProperty().bind(this.styleableHeight);
    }


    // ------------------------------------------------------------------------
    // ---------------------- PROPERTY METHODS --------------------------------
    // ------------------------------------------------------------------------

    public final double getStyleableHeight() {
        return this.styleableHeight.get();
    }

    public final void setStyleableHeight(double value) {
        this.styleableHeight.set(value);
    }

    public final DoubleProperty styleableHeightProperty() {
        return this.styleableHeight;
    }

    public final double getStyleableWidth() {
        return this.styleableWidth.get();
```

```
    }

    public final void setStyleableWidth(double value) {
        this.styleableWidth.set(value);
    }

    public final DoubleProperty styleableWidthProperty() {
        return this.styleableWidth;
    }

}
```

Прочитайте CSS онлайн: https://riptutorial.com/ru/javafx/topic/1581/css

глава 3: FXML и контроллеры

Синтаксис

- xmlns: fx = " http://javafx.com/fxml " // Объявление пространства имен

Examples

Пример FXML

Простой документ FXML, в котором описывается `AnchorPane` содержащая кнопку и узел метки:

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

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
xmlns:fx="http://javafx.com/fxml/1"
        fx:controller="com.example.FXMLDocumentController">
    <children>
        <Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleButtonAction"
fx:id="button" />
        <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="label" />
    </children>
</AnchorPane>
```

Этот пример файла FXML связан с классом контроллера. Связь между FXML и классом контроллера в этом случае производится путем указания имени класса как значения атрибута `fx:controller` в корневом элементе FXML: `fx:controller="com.example.FXMLDocumentController"` . Класс контроллера позволяет выполнять Java-код в ответ на действия пользователя на элементах пользовательского интерфейса, определенных в файле FXML:

```
package com.example ;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;

public class FXMLDocumentController {

    @FXML
```

```
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle resources) {
        // Initialization code can go here.
        // The parameters url and resources can be omitted if they are not needed
    }

}
```

`FXMLLoader` можно использовать для загрузки файла FXML:

```
public class MyApp extends Application {

    @Override
    public void start(Stage stage) throws Exception {

        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource("FXMLDocument.fxml"));
        Parent root = loader.load();

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }

}
```

Метод `load` выполняет несколько действий, и полезно понять порядок их возникновения. В этом простом примере:

1. `FXMLLoader` считывает и анализирует файл FXML. Он создает объекты, соответствующие элементам, определенным в файле, и учитывает любые атрибуты `fx:id` определенные на них.

2. Поскольку корневой элемент файла FXML определяет атрибут `fx:controller` , `FXMLLoader` создает *новый экземпляр* класса, который он задает. По умолчанию это происходит при вызове конструктора без аргументов в указанном классе.

3. Любые элементы с определенными атрибутами `fx:id` которые имеют поля в контроллере с соответствующими именами полей и которые являются либо `public` (не рекомендуется), либо аннотированными `@FXML` (рекомендуется), вводятся в соответствующие поля. Итак, в этом примере, поскольку в файле FXML есть `Label` с `fx:id="label"` и поле в контроллере, определенное как

   ```
   @FXML
   ```

```
private Label label ;
```

поле `label` инициализируется экземпляром `Label` созданным `FXMLLoader` .

4. Обработчики событий регистрируются любыми элементами в файле FXML с установленными свойствами onXXX `onXXX="#..."` . Эти обработчики событий вызывают указанный метод в классе контроллера. В этом примере, поскольку `Button` имеет `onAction="#handleButtonAction"` , а контроллер определяет метод

```
@FXML
private void handleButtonAction(ActionEvent event) { ... }
```

когда действие запускается над кнопкой (например, пользователь нажимает на нее), этот метод вызывается. Метод должен иметь тип возврата `void` и может либо определять параметр, соответствующий типу события (`ActionEvent` в этом примере), либо не определять параметры.

5. Наконец, если класс контроллера определяет метод `initialize` , этот метод вызывается. Обратите внимание, что это происходит после `@FXML` полей `@FXML` , поэтому они могут быть безопасно доступны в этом методе и будут инициализированы экземплярами, соответствующими элементам в файле FXML. Метод `initialize()` может либо не принимать никаких параметров, либо принимать `URL` и `ResourceBundle` . В последнем случае эти параметры будут заполнены `URL` представляющим местоположение файла FXML, и любым `ResourceBundle` установленным на `FXMLLoader` через `loader.setResources(...)` . Любой из них может быть `null` если они не были установлены.

Вложенные контроллеры

Нет необходимости создавать весь пользовательский интерфейс в одном FXML, используя один контроллер.

Тег `<fx:include>` можно использовать для включения одного файла fxml в другой. Контроллер включенного fxml может быть введен в контроллер включенного файла так же, как и любой другой объект, созданный `FXMLLoader` .

Это делается добавлением атрибута `fx:id` в элемент `<fx:include>` . Таким образом, контроллер включенного fxml будет введен в поле с именем `<fx:id value>Controller` .

Примеры:

Значение fx: id	название поля для инъекций
Foo	FooController
answer42	answer42Controller

Значение fx: id	название поля для инъекций
АБВ	xYzController

Образец fxmls

счетчик

Это fxml, содержащий `StackPane` с `Text` узлом. Контроллер для этого файла fxml позволяет получать текущее значение счетчика, а также увеличивать счетчик:

counter.fxml

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

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

<StackPane prefHeight="200" prefWidth="200" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="counter.CounterController">
    <children>
        <Text fx:id="counter" />
    </children>
</StackPane>
```

CounterController

```
package counter;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class CounterController {
    @FXML
    private Text counter;

    private int value = 0;

    public void initialize() {
        counter.setText(Integer.toString(value));
    }

    public void increment() {
        value++;
        counter.setText(Integer.toString(value));
    }

    public int getValue() {
        return value;
    }

}
```

Включая fxml

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

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

<BorderPane prefHeight="500" prefWidth="500" xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="counter.OuterController">
    <left>
        <Button BorderPane.alignment="CENTER" text="increment" onAction="#increment" />
    </left>
    <center>
        <!-- content from counter.fxml included here -->
        <fx:include fx:id="count" source="counter.fxml" />
    </center>
</BorderPane>
```

OuterController

Контроллер включенного fxml вводится в этот контроллер. Здесь обработчик для события `onAction` для `Button` используется для увеличения счетчика.

```
package counter;

import javafx.fxml.FXML;

public class OuterController {

    // controller of counter.fxml injected here
    @FXML
    private CounterController countController;

    public void initialize() {
        // controller available in initialize method
        System.out.println("Current value: " + countController.getValue());
    }

    @FXML
    private void increment() {
        countController.increment();
    }

}
```

Fxmls можно загрузить таким образом, предполагая, что код вызывается из класса в том же пакете, что и `outer.fxml` :

```
Parent parent = FXMLLoader.load(getClass().getResource("outer.fxml"));
```

Определить блоки и

Иногда элемент должен быть создан за пределами обычной структуры объекта в fxml.

Здесь *Define Blocks* вступают в игру:

Содержимое внутри элемента `<fx:define>` не добавляется к объекту, созданному для

родительского элемента.

Каждому дочернему элементу `<fx:define>` нужен атрибут `fx:id` .

Объекты, созданные таким образом, могут быть позже указаны с помощью элемента `<fx:reference>` или с помощью привязки выражения.

Элемент `<fx:reference>` может использоваться для ссылки на любой элемент с атрибутом `fx:id` который обрабатывается до обработки элемента `<fx:reference>` , используя то же значение, что и атрибут `fx:id` ссылочного элемента в `source` атрибут элемента `<fx:reference>` .

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

<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>


<VBox xmlns:fx="http://javafx.com/fxml/1" prefHeight="300.0" prefWidth="300.0"
xmlns="http://javafx.com/javafx/8">
    <children>
        <fx:define>
            <String fx:value="My radio group" fx:id="text" />
        </fx:define>
        <Text>
            <text>
                <!-- reference text defined above using fx:reference -->
                <fx:reference source="text"/>
            </text>
        </Text>
        <RadioButton text="Radio 1">
            <toggleGroup>
                <ToggleGroup fx:id="group" />
            </toggleGroup>
        </RadioButton>
        <RadioButton text="Radio 2">
            <toggleGroup>
                <!-- reference ToggleGroup created for last RadioButton -->
                <fx:reference source="group"/>
            </toggleGroup>
        </RadioButton>
        <RadioButton text="Radio 3" toggleGroup="$group" />

        <!-- reference text defined above using expression binding -->
        <Text text="$text" />
    </children>
</VBox>
```

Передача данных в FXML - доступ к существующему контроллеру

Проблема. Некоторые данные должны быть переданы в сцену, загруженную из fxml.

Решение

Укажите контроллер, используя атрибут `fx:controller` и получите экземпляр контроллера, созданный во время процесса загрузки, из экземпляра `FXMLLoader` используемого для загрузки fxml.

Добавьте методы для передачи данных в экземпляр контроллера и обработайте данные в этих методах.

FXML

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

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

<VBox xmlns:fx="http://javafx.com/fxml/1" fx:controller="valuepassing.TestController">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

контроллер

```
package valuepassing;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    @FXML
    private Text target;

    public void setData(String data) {
        target.setText(data);
    }

}
```

Код, используемый для загрузки fxml

```
String data = "Hello World!";

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));
Parent root = loader.load();
TestController controller = loader.<TestController>getController();
controller.setData(data);
```

Передача данных в FXML - указание экземпляра контроллера

Проблема. Некоторые данные должны быть переданы в сцену, загруженную из fxml.

Решение

Установите контроллер с `FXMLLoader` экземпляра `FXMLLoader` используется позже для загрузки fxml.

Перед загрузкой fxml убедитесь, что контроллер содержит соответствующие данные.

Примечание: в этом случае файл fxml не должен содержать атрибут `fx:controller` .

FXML

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

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

<VBox xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

контроллер

```
import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    private final String data;

    public TestController(String data) {
        this.data = data;
    }

    @FXML
    private Text target;

    public void initialize() {
        // handle data once the fields are injected
        target.setText(data);
    }

}
```

Код, используемый для загрузки fxml

```
String data = "Hello World!";

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));

TestController controller = new TestController(data);
loader.setController(controller);

Parent root = loader.load();
```

Передача параметров в FXML - с использованием контроллераFactory

Проблема. Некоторые данные должны быть переданы в сцену, загруженную из fxml.

Решение

Укажите фабрику контроллера, которая отвечает за создание контроллеров. Передайте данные экземпляру контроллера, созданному на заводе.

FXML

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

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

<VBox xmlns:fx="http://javafx.com/fxml/1" fx:controller="valuepassing.TestController">
    <children>
        <Text fx:id="target" />
    </children>
</VBox>
```

контроллер

```
package valuepassing;

import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class TestController {

    private final String data;

    public TestController(String data) {
        this.data = data;
    }

    @FXML
    private Text target;

    public void initialize() {
        // handle data once the fields are injected
        target.setText(data);
    }

}
```

Код, используемый для загрузки fxml

Строковые данные = «Hello World!»;

```
Map<Class, Callable<?>> creators = new HashMap<>();
creators.put(TestController.class, new Callable<TestController>() {

    @Override
```

```
    public TestController call() throws Exception {
        return new TestController(data);
    }

});

FXMLLoader loader = new FXMLLoader(getClass().getResource("test.fxml"));

loader.setControllerFactory(new Callback<Class<?>, Object>() {

    @Override
    public Object call(Class<?> param) {
        Callable<?> callable = creators.get(param);
        if (callable == null) {
            try {
                // default handling: use no-arg constructor
                return param.newInstance();
            } catch (InstantiationException | IllegalAccessException ex) {
                throw new IllegalStateException(ex);
            }
        } else {
            try {
                return callable.call();
            } catch (Exception ex) {
                throw new IllegalStateException(ex);
            }
        }
    }
});

Parent root = loader.load();
```

Это может показаться сложным, но может быть полезно, если fxml должен решить, какой класс контроллера ему нужен.

Создание экземпляра в FXML

Следующий класс используется для демонстрации того, как могут создаваться экземпляры классов:

JavaFX 8

Аннотацию в `Person(@NamedArg("name") String name)` необходимо удалить, поскольку аннотация `@NamedArg` недоступна.

```
package fxml.sample;

import javafx.beans.NamedArg;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Person {

    public static final Person JOHN = new Person("John");
```

```
    public Person() {
        System.out.println("Person()");
    }

    public Person(@NamedArg("name") String name) {
        System.out.println("Person(String)");
        this.name.set(name);
    }

    public Person(Person person) {
        System.out.println("Person(Person)");
        this.name.set(person.getName());
    }

    private final StringProperty name = new SimpleStringProperty();

    public final String getName() {
        System.out.println("getter");
        return this.name.get();
    }

    public final void setName(String value) {
        System.out.println("setter");
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        System.out.println("property getter");
        return this.name;
    }

    public static Person valueOf(String value) {
        System.out.println("valueOf");
        return new Person(value);
    }

    public static Person createPerson() {
        System.out.println("createPerson");
        return new Person();
    }

}
```

Предположим, что класс `Person` уже был инициализирован перед загрузкой fxml.

Примечание об импорте

В следующем примере fxml раздел импорта будет отсутствовать. Однако fxml должен начинаться с

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

а затем раздел импорта, импортирующий все классы, используемые в файле fxml. Импорт аналогичен нестационарному импорту, но добавляется в качестве инструкций по обработке. **Необходимо импортировать даже классы из пакета `java.lang` .**

В этом случае следует добавить следующие импорты:

```
<?import java.lang.*?>
<?import fxml.sample.Person?>
```

JavaFX 8

Аннотированный конструктор `@NamedArg`

Если есть конструктор, где каждый параметр аннотируется с `@NamedArg` и все значения аннотаций `@NamedArg` присутствуют в fxml, конструктор будет использоваться с этими параметрами.

```
<Person name="John"/>
```

```
<Person xmlns:fx="http://javafx.com/fxml">
    <name>
        <String fx:value="John"/>
    </name>
</Person>
```

Оба они приводят к следующему выходу консоли, если они загружаются:

```
Person(String)
```

Нет конструктора args

Если нет подходящего `@NamedArg` annotated `@NamedArg` будет использоваться конструктор, который не принимает никаких параметров.

Удалите аннотацию `@NamedArg` из конструктора и попробуйте загрузить.

```
<Person name="John"/>
```

Это будет использовать конструктор без параметров.

Выход:

```
Person()
setter
```

атрибут `fx:value`

Атрибут `fx:value` может использоваться для передачи его значения `static` методу `valueOf` принимающему параметр `String` и возвращающему экземпляр для использования.

пример

```
<Person xmlns:fx="http://javafx.com/fxml" fx:value="John"/>
```

Выход:

```
valueOf
Person(String)
```

fx:factory

Атрибут `fx:factory` позволяет создавать объекты с использованием произвольных `static` методов, которые не принимают параметры.

пример

```
<Person xmlns:fx="http://javafx.com/fxml" fx:factory="createPerson">
    <name>
        <String fx:value="John"/>
    </name>
</Person>
```

Выход:

```
createPerson
Person()
setter
```

<fx:copy>

С помощью `fx:copy` copy можно вызвать конструктор копирования. Указание `fx:id` другого. `source` атрибут тега будет вызывать конструктор копирования с этим объектом в качестве параметра.

Пример:

```
<ArrayList xmlns:fx="http://javafx.com/fxml">
    <Person fx:id="p1" fx:constant="JOHN"/>
    <fx:copy source="p1"/>
</ArrayList>
```

Выход

```
Person(Person)
getter
```

fx:constant

`fx:constant` позволяет получить значение из `static final` поля.

пример

```
<Person xmlns:fx="http://javafx.com/fxml" fx:constant="JOHN"/>
```

не будет выдавать какой-либо вывод, так как это просто ссылается на `JOHN` который был создан при инициализации класса.

Настройка свойств

Существует несколько способов добавления данных в объект в файле fxml:

`<property>`

Тег с именем свойства может быть добавлен как дочерний элемент элемента, используемого для создания экземпляра. Ребенку этого тега присваивается свойство с помощью установщика или добавляется к содержимому свойства (свойства только для чтения / карты).

Свойство по умолчанию

Класс можно аннотировать с `@DefaultProperty` аннотации `@DefaultProperty` . В этом случае элементы могут быть непосредственно добавлены как дочерний элемент без использования элемента с именем свойства.

`property="value"`

Свойства могут быть назначены с использованием имени свойства в качестве имени атрибута и значения как значения атрибута. Это имеет тот же эффект, что и добавление следующего элемента в качестве дочернего элемента тега:

```
<property>
    <String fx:value="value" />
</property>
```

статические сеттеры

Свойства можно также установить с помощью `static` сеттеров. Это `static` методы с именем `setProperty` которые принимают элемент как первый параметр и значение, заданное как второй параметр. Эти методы могут указывать в любом классе и могут использоваться с использованием `ContainingClass.property` вместо обычного имени свойства.

Примечание. В настоящее время кажется необходимым иметь соответствующий статический метод getter (то есть статический метод с именем `getProperty` принимает

элемент как параметр в том же классе, что и статический сеттер), чтобы это работало, если только тип значения не является `String` .

Тип Принуждение

Следующий механизм используется для получения объекта правильного класса во время присвоений, например, для соответствия типу параметра метода сеттера.

Если классы назначаются, то используется само значение.

В противном случае значение преобразуется следующим образом

Тип цели	значение используется (значение источника `s`)
`Boolean` , `boolean`	`Boolean.valueOf(s)`
`char` , `Character`	`s.toString.charAt(0)`
другой примитивный тип или тип обертки	соответствующий метод для типа цели, в случае, когда `s` является `Number` , значение `valueOf(s.toString())` для типа обертки иначе
`BigInteger`	`BigInteger.valueOf(s.longValue())` является `s` это `Number` , `new BigInteger(s.toString())` в противном случае
`BigDecimal`	`BigDecimal.valueOf(s.doubleValue())` является `s` это `Number` , `new BigDecimal(s.toString())` в противном случае
Число	`Double.valueOf(s.toString())` если `s.toString()` содержит а `.` , `Long.valueOf(s.toString())` противном случае
`Class`	`Class.forName(s.toString())` вызванный с использованием контекста `ClassLoader` текущего потока без инициализации класса
перечисление	Результат метода `valueOf` , дополнительно преобразованный во всю строчную `String` разделенную _ вставленную перед каждой прописной буквой, если `s` является `String` которая начинается с буквы в нижнем регистре
Другой	значение, возвращаемое `static` методом `valueOf` в targetType, которое имеет параметр, соответствующий типу `s` или суперклассу этого типа

Примечание. Это поведение плохо документировано и может быть изменено.

пример

```
public enum Location {
    WASHINGTON_DC,
    LONDON;
}
```

```
package fxml.sample;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javafx.beans.DefaultProperty;

@DefaultProperty("items")
public class Sample {

    private Location loaction;

    public Location getLoaction() {
        return loaction;
    }

    public void setLoaction(Location loaction) {
        this.loaction = loaction;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    int number;

    private final List<Object> items = new ArrayList<>();

    public List<Object> getItems() {
        return items;
    }

    private final Map<String, Object> map = new HashMap<>();

    public Map<String, Object> getMap() {
        return map;
    }

    private BigInteger serialNumber;

    public BigInteger getSerialNumber() {
        return serialNumber;
    }

    public void setSerialNumber(BigInteger serialNumber) {
```

```
        this.serialNumber = serialNumber;
    }

    @Override
    public String toString() {
        return "Sample{" + "loaction=" + loaction + ", number=" + number + ", items=" + items
+ ", map=" + map + ", serialNumber=" + serialNumber + '}';
    }

}
```

```
package fxml.sample;

public class Container {

    public static int getNumber(Sample sample) {
        return sample.number;
    }

    public static void setNumber(Sample sample, int number) {
        sample.number = number;
    }

    private final String value;

    private Container(String value) {
        this.value = value;
    }

    public static Container valueOf(String s) {
        return new Container(s);
    }

    @Override
    public String toString() {
        return "42" + value;
    }

}
```

Печать результата загрузки ниже `fxml` файлов `fxml`

```
Sample{loaction=WASHINGTON_DC, number=5, items=[42a, 42b, 42c, 42d, 42e, 42f], map={answer=42,
g=9.81, hello=42A, sample=Sample{loaction=null, number=33, items=[], map={},
serialNumber=null}}, serialNumber=4299}
```

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

<?import java.lang.*?>
<?import fxml.sample.*?>

<Sample xmlns:fx="http://javafx.com/fxml/1" Container.number="5" loaction="washingtonDc">

    <!-- set serialNumber property (type coercion) -->
    <serialNumber>
        <Container fx:value="99"/>
    </serialNumber>
```

```
    <!-- Add elements to default property-->
    <Container fx:value="a"/>
    <Container fx:value="b"/>
    <Container fx:value="c"/>
    <Container fx:value="d"/>
    <Container fx:value="e"/>
    <Container fx:value="f"/>

    <!-- fill readonly map property -->
    <map g="9.81">
        <hello>
            <Container fx:value="A"/>
        </hello>
        <answer>
            <Container fx:value=""/>
        </answer>
        <sample>
            <Sample>
                <!-- static setter-->
                <Container.number>
                    <Integer fx:value="33" />
                </Container.number>
            </Sample>
        </sample>
    </map>
</Sample>
```

Прочитайте FXML и контроллеры онлайн: https://riptutorial.com/ru/javafx/topic/1580/fxml-и-контроллеры

глава 4: ScrollPane

Вступление

ScrollPane - это элемент управления, который предлагает динамическое представление его содержимого. Этот взгляд контролируется по-разному; (кнопка увеличения / уменьшения / колесико мыши), чтобы иметь встроенный вид содержимого.

Examples

A) Размер фиксированного контента:

Размер содержимого будет таким же, как размер его контейнера ScrollPane.

```
import javafx.scene.control.ScrollPane;   //Import the ScrollPane
import javafx.scene.control.ScrollPane.ScrollBarPolicy; //Import the ScrollBarPolicy
import javafx.scene.layout.Pane;

ScrollPane scrollpane;
Pane content = new Pane();  //We will use this Pane as a content

scrollpane = new ScrollPane(content);  //Initialize and add content as a parameter
scrollpane.setPrefSize(300, 300);   //Initialize the size of the ScrollPane

scrollpane.setFitToWidth(true);  //Adapt the content to the width of ScrollPane
scrollpane.setFitToHeight(true); //Adapt the content to the height of ScrollPane


scrollpane.setHbarPolicy(ScrollBarPolicy.ALWAYS);  //Control the visibility of the Horizontal
ScrollBar
scrollpane.setVbarPolicy(ScrollBarPolicy.NEVER);  //Control the visibility of the Vertical
ScrollBar
//There are three types of visibility (ALWAYS/AS_NEEDED/NEVER)
```

B) Размер динамического содержимого:

Размер содержимого будет изменяться в зависимости от добавленных элементов, которые превышают пределы содержания в обеих осях (горизонтальные и вертикальные), которые можно увидеть, перемещаясь по представлению.

```
import javafx.scene.control.ScrollPane;   //Import the ScrollPane
import javafx.scene.control.ScrollPane.ScrollBarPolicy; //Import the ScrollBarPolicy
import javafx.scene.layout.Pane;

ScrollPane scrollpane;
Pane content = new Pane();  //We will use this Pane as a content

scrollpane = new ScrollPane();
scrollpane.setPrefSize(300, 300);   //Initialize the size of the ScrollPane
```

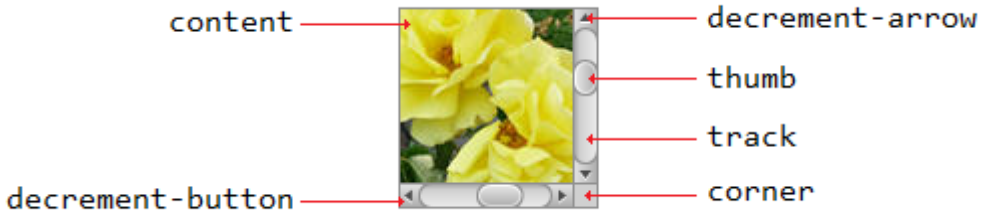
```
content.setMinSize(300,300); //Here a minimum size is set so that the container can be
extended.
scrollpane.setContent(content); // we add the content to the ScrollPane
```

Примечание. Здесь нам не нужны оба метода (setFitToWidth / setFitToHeight).

Стилизация ScrollPane:

Появление ScrollPane можно легко изменить, имея некоторые понятия « *CSS* » и уважая некоторые « *свойства* » контроля и, конечно же, обладающие некоторой « *фантазией* ».

A) Элементы, составляющие ScrollPane:



B) Свойства CSS:

```
.scroll-bar:vertical .track{}

.scroll-bar:horizontal .track{}

.scroll-bar:horizontal .thumb{}

.scroll-bar:vertical .thumb{}


.scroll-bar:vertical *.increment-button,
.scroll-bar:vertical *.decrement-button{}

.scroll-bar:vertical *.increment-arrow .content,
.scroll-bar:vertical *.decrement-arrow .content{}

.scroll-bar:vertical *.increment-arrow,
.scroll-bar:vertical *.decrement-arrow{}


.scroll-bar:horizontal *.increment-button,
.scroll-bar:horizontal *.decrement-button{}

.scroll-bar:horizontal *.increment-arrow .content,
.scroll-bar:horizontal *.decrement-arrow .content{}

.scroll-bar:horizontal *.increment-arrow,
.scroll-bar:horizontal *.decrement-arrow{}

.scroll-pane .corner{}

.scroll-pane{}
```

Прочитайте ScrollPane онлайн: https://riptutorial.com/ru/javafx/topic/8259/scrollpane

глава 5: TableView

Examples

Пример TableView с двумя столбцами

Таблица

Следующий класс содержит 2 свойства: имя (`String`) и размер (`double`). Оба свойства завернуты в свойства JavaFX, чтобы `TableView` мог наблюдать изменения.

```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Person {

    public Person(String name, double size) {
        this.size = new SimpleDoubleProperty(this, "size", size);
        this.name = new SimpleStringProperty(this, "name", name);
    }

    private final StringProperty name;
    private final DoubleProperty size;

    public final String getName() {
        return this.name.get();
    }

    public final void setName(String value) {
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        return this.name;
    }

    public final double getSize() {
        return this.size.get();
    }

    public final void setSize(double value) {
        this.size.set(value);
    }

    public final DoubleProperty sizeProperty() {
        return this.size;
    }

}
```

Пример приложения

Это приложение отображает `TableView` с двумя столбцами; один для имени и один для размера `Person` . Выбор одного из `Person` s добавляет данные в `TextField` s ниже `TableView` и позволяет пользователю редактировать данные. Обратите внимание, как только редактирование завершено, `TableView` автоматически обновляется.

Каждому для каждого `TableColumn` добавленному в `TableView` , присваивается `cellValueFactory` . Эта фабрика отвечает за преобразование элементов таблицы (`Person` 's) в `ObservableValue` s, которые содержат значение, которое должно отображаться в ячейке таблицы, и которое позволяет `TableView` прослушивать любые изменения для этого значения.

```
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.TextFormatter;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;
import javafx.util.StringConverter;

public class TableSample extends Application {

    @Override
    public void start(Stage primaryStage) {
        // data for the tableview. modifying this list automatically updates the tableview
        ObservableList<Person> data = FXCollections.observableArrayList(
                new Person("John Doe", 1.75),
                new Person("Mary Miller", 1.70),
                new Person("Frank Smith", 1.80),
                new Person("Charlotte Hoffman", 1.80)
        );

        TableView<Person> tableView = new TableView<>(data);

        // table column for the name of the person
        TableColumn<Person, String> nameColumn = new TableColumn<>("Name");
        nameColumn.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Person,
String>, ObservableValue<String>>() {

            @Override
            public ObservableValue<String> call(TableColumn.CellDataFeatures<Person, String>
param) {
                return param.getValue().nameProperty();
            }
        });

        // column for the size of the person
```

```
        TableColumn<Person, Number> sizeColumn = new TableColumn<>("Size");
        sizeColumn.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Person,
Number>, ObservableValue<Number>>() {

            @Override
            public ObservableValue<Number> call(TableColumn.CellDataFeatures<Person, Number>
param) {
                return param.getValue().sizeProperty();
            }
        });

        // add columns to tableview
        tableView.getColumns().addAll(nameColumn, sizeColumn);

        TextField name = new TextField();

        TextField size = new TextField();

        // convert input from textfield to double
        TextFormatter<Double> sizeFormatter = new TextFormatter<Double>(new
StringConverter<Double>() {

            @Override
            public String toString(Double object) {
                return object == null ? "" : object.toString();
            }

            @Override
            public Double fromString(String string) {
                if (string == null || string.isEmpty()) {
                    return null;
                } else {
                    try {
                        double val = Double.parseDouble(string);
                        return val < 0 ? null : val;
                    } catch (NumberFormatException ex) {
                        return null;
                    }
                }
            }

        });
        size.setTextFormatter(sizeFormatter);

        Button commit = new Button("Change Item");
        commit.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                Person p = tableView.getSelectionModel().getSelectedItem();
                p.setName(name.getText());
                Double value = sizeFormatter.getValue();
                p.setSize(value == null ? -1d : value);
            }

        });

        // listen for changes in the selection to update the data in the textfields
        tableView.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<Person>() {
```

```
            @Override
            public void changed(ObservableValue<? extends Person> observable, Person oldValue,
Person newValue) {
                commit.setDisable(newValue == null);
                if (newValue != null) {
                    sizeFormatter.setValue(newValue.getSize());
                    name.setText(newValue.getName());
                }
            }

        });

        HBox editors = new HBox(5, new Label("Name:"), name, new Label("Size: "), size,
commit);

        VBox root = new VBox(10, tableView, editors);

        Scene scene = new Scene(root);

        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

PropertyValueFactory

`PropertyValueFactory` может использоваться как `cellValueFactory` в `TableColumn` . Он использует отражение для доступа к методам, которые соответствуют определенному шаблону для извлечения данных из элемента `TableView` :

пример

```
TableColumn<Person, String> nameColumn = ...
PropertyValueFactory<Person, String> valueFactory = new PropertyValueFactory<>("name");
nameColumn.setCellValueFactory(valueFactory);
```

Имя метода, используемого для получения данных, зависит от параметра-параметра конструктора для `PropertyValueFactory` .

- **Метод свойства:** ожидается, что этот вид метода возвращает `ObservableValue` содержащий данные. Могут наблюдаться изменения. Они должны соответствовать шаблону `<constructor parameter>Property` и не принимать никаких параметров.
- **Метод Getter:** этот метод предполагает возврат значения напрямую (`String` в приведенном выше примере). Имя метода должно соответствовать шаблону `get<Constructor parameter>` . Обратите внимание, что здесь `<Constructor parameter>` начинается с *прописной буквы* . Этот метод не должен принимать параметры.

Примеры имен методов

параметр конструктора (без кавычек)	название метода собственности	имя метода геттера
Foo	fooProperty	getFoo
Foobar	fooBarProperty	getFooBar
XYZ	XYZProperty	getXYZ
ListIndex	listIndexProperty	getListIndex
ценность	aValueProperty	getAValue

Настройка TableCell выглядит в зависимости от элемента

Иногда в столбце должен отображаться другой контент, чем только значение `toString` элемента ячейки. В этом случае `TableCell` s, созданный `cellFactory` `TableColumn` , настроен для изменения макета на основе элемента.

Важное примечание. `TableView` создает только `TableCell` s, которые показаны в пользовательском интерфейсе. Элементы внутри ячеек могут меняться и даже становиться пустыми. Программисту необходимо позаботиться о том, чтобы отменить любые изменения в `TableCell` которые были сделаны, когда элемент был добавлен при его удалении. В противном случае содержимое может отображаться в ячейке, где «она не принадлежит».

В приведенном ниже примере установка элемента приводит к тому, что текст будет установлен, а также изображение, отображаемое в `ImageView` :

```
image.setImage(item.getEmoji());
setText(item.getValue());
```

Если элемент становится `null` или ячейка становится пустой, эти изменения отменены, если значения вернутся к `null` значению:

```
setText(null);
image.setImage(null);
```

Следующий пример показывает emoji в дополнение к тексту в `TableCell` .

Метод `updateItem` вызывается каждый раз, когда элемент `Cell` изменяется. Переопределение этого метода позволяет реагировать на изменения и корректировать внешний вид ячейки. Добавление слушателя к `itemProperty()` ячейки было бы альтернативой, но во многих случаях `TableCell` расширяется.

Тип элемента

```
import javafx.scene.image.Image;

// enum providing image and text for certain feelings
public enum Feeling {
    HAPPY("happy",
"https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/Emojione_1F600.svg/64px-
Emojione_1F600.svg.png"),
    SAD("sad",
"https://upload.wikimedia.org/wikipedia/commons/thumb/4/42/Emojione_1F62D.svg/64px-
Emojione_1F62D.svg.png")
    ;
    private final Image emoji;
    private final String value;

    Feeling(String value, String url) {
        // load image in background
        emoji = new Image(url, true);
        this.value = value;
    }

    public Image getEmoji() {
        return emoji;
    }

    public String getValue() {
        return value;
    }

}
```

Код в классе приложения

```
import javafx.application.Application;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;

public class EmotionTable extends Application {

    public static class Item {

        private final ObjectProperty<Feeling> feeling;

        public Item(Feeling feeling) {
            this.feeling = new SimpleObjectProperty<>(feeling);
        }
```

```
        public final Feeling getFeeling() {
            return this.feeling.get();
        }

        public final void setFeeling(Feeling value) {
            this.feeling.set(value);
        }

        public final ObjectProperty<Feeling> feelingProperty() {
            return this.feeling;
        }

    }

    @Override
    public void start(Stage primaryStage) {
        TableView<Item> table = new TableView<>(FXCollections.observableArrayList(
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.SAD),
                null,
                new Item(Feeling.HAPPY),
                new Item(Feeling.HAPPY),
                new Item(Feeling.SAD)
        ));

        EventHandler<ActionEvent> eventHandler = new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                // change table items depending on userdata of source
                Node source = (Node) event.getSource();
                Feeling targetFeeling = (Feeling) source.getUserData();
                for (Item item : table.getItems()) {
                    if (item != null) {
                        item.setFeeling(targetFeeling);
                    }
                }
            }

        };

        TableColumn<Item, Feeling> feelingColumn = new TableColumn<>("Feeling");

        feelingColumn.setCellValueFactory(new PropertyValueFactory<>("feeling"));

        // use custom tablecell to display emoji image
        feelingColumn.setCellFactory(new Callback<TableColumn<Item, Feeling>, TableCell<Item,
Feeling>>() {

            @Override
            public TableCell<Item, Feeling> call(TableColumn<Item, Feeling> param) {
                return new EmojiCell<>();
            }
        });

        table.getColumns().add(feelingColumn);

        Button sunshine = new Button("sunshine");
```

```
        Button rain = new Button("rain");

        sunshine.setOnAction(eventHandler);
        rain.setOnAction(eventHandler);

        sunshine.setUserData(Feeling.HAPPY);
        rain.setUserData(Feeling.SAD);

        Scene scene = new Scene(new VBox(10, table, new HBox(10, sunshine, rain)));

        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

Класс ячейки

```
import javafx.scene.control.TableCell;
import javafx.scene.image.ImageView;

public class EmojiCell<T> extends TableCell<T, Feeling> {

    private final ImageView image;

    public EmojiCell() {
        // add ImageView as graphic to display it in addition
        // to the text in the cell
        image = new ImageView();
        image.setFitWidth(64);
        image.setFitHeight(64);
        image.setPreserveRatio(true);

        setGraphic(image);
        setMinHeight(70);
    }

    @Override
    protected void updateItem(Feeling item, boolean empty) {
        super.updateItem(item, empty);

        if (empty || item == null) {
            // set back to look of empty cell
            setText(null);
            image.setImage(null);
        } else {
            // set image and text for non-empty cell
            image.setImage(item.getEmoji());
            setText(item.getValue());
        }
    }
}
```

Добавить кнопку в Tableview

Вы можете добавить кнопку или другой компонент javafx в Tableview, используя `setCellFactory(Callback value)` .

Пример приложения

В этом приложении мы добавим кнопку в TableView. При нажатии на эту кнопку столбца выбираются данные в той же строке, что и кнопка, и распечатывается ее информация.

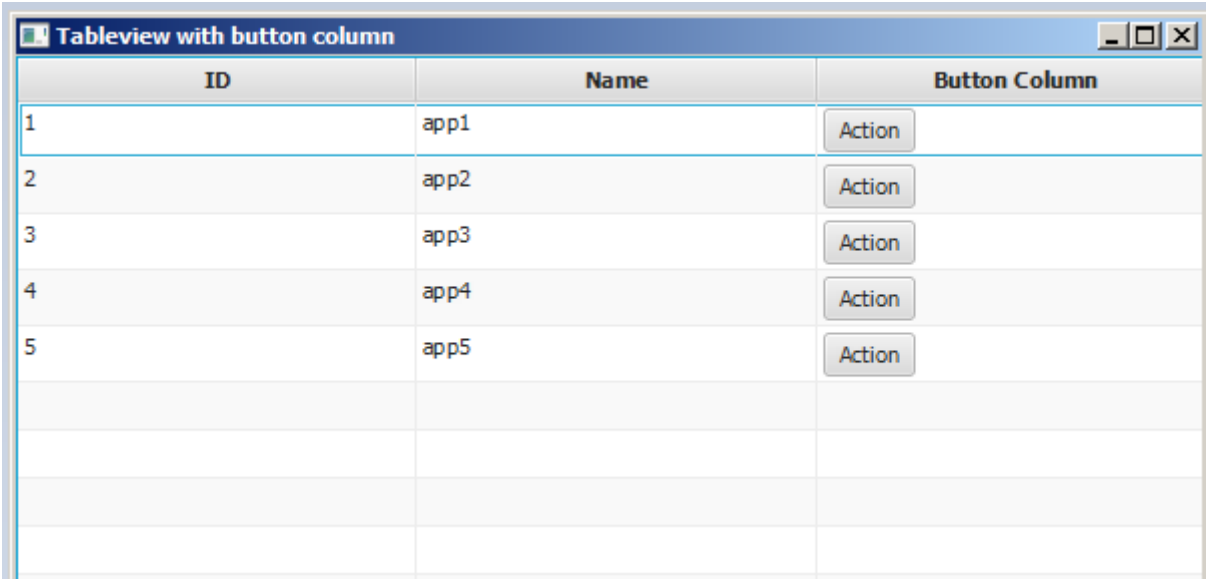
В `addButtonToTable()` `cellFactory` вызов `cellFactory` отвечает за добавление кнопки в соответствующий столбец. Мы определяем вызываемый cellFactory и реализуем его метод override `call(...)` чтобы получить `TableCell` с кнопкой, а затем этот набор `cellFactory` установлен в соответствующий `setCellFactory(..)` . В нашем примере это `colBtn.setCellFactory(cellFactory)` . SSCCE ниже:

```
import javafx.application.Application;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import javafx.util.Callback;

public class TableViewSample extends Application {

    private final TableView<Data> table = new TableView<>();
    private final ObservableList<Data> tvObservableList = FXCollections.observableArrayList();

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {

        stage.setTitle("Tableview with button column");
        stage.setWidth(600);
        stage.setHeight(600);

        setTableappearance();

        fillTableObservableListWithSampleData();
        table.setItems(tvObservableList);

        TableColumn<Data, Integer> colId = new TableColumn<>("ID");
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));

        TableColumn<Data, String> colName = new TableColumn<>("Name");
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));
```

```
        table.getColumns().addAll(colId, colName);

        addButtonToTable();

        Scene scene = new Scene(new Group(table));

        stage.setScene(scene);
        stage.show();
    }

    private void setTableappearance() {
        table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        table.setPrefWidth(600);
        table.setPrefHeight(600);
    }

    private void fillTableObservableListWithSampleData() {

        tvObservableList.addAll(new Data(1, "app1"),
                                new Data(2, "app2"),
                                new Data(3, "app3"),
                                new Data(4, "app4"),
                                new Data(5, "app5"));
    }

    private void addButtonToTable() {
        TableColumn<Data, Void> colBtn = new TableColumn("Button Column");

        Callback<TableColumn<Data, Void>, TableCell<Data, Void>> cellFactory = new
Callback<TableColumn<Data, Void>, TableCell<Data, Void>>() {
            @Override
            public TableCell<Data, Void> call(final TableColumn<Data, Void> param) {
                final TableCell<Data, Void> cell = new TableCell<Data, Void>() {

                    private final Button btn = new Button("Action");

                    {
                        btn.setOnAction((ActionEvent event) -> {
                            Data data = getTableView().getItems().get(getIndex());
                            System.out.println("selectedData: " + data);
                        });
                    }

                    @Override
                    public void updateItem(Void item, boolean empty) {
                        super.updateItem(item, empty);
                        if (empty) {
                            setGraphic(null);
                        } else {
                            setGraphic(btn);
                        }
                    }
                };
                return cell;
            }
        };

        colBtn.setCellFactory(cellFactory);

        table.getColumns().add(colBtn);
```

```
    }

    public class Data {

        private int id;
        private String name;

        private Data(int id, String name) {
            this.id = id;
            this.name = name;
        }

        public int getId() {
            return id;
        }

        public void setId(int ID) {
            this.id = ID;
        }

        public String getName() {
            return name;
        }

        public void setName(String nme) {
            this.name = nme;
        }

        @Override
        public String toString() {
            return "id: " + id + " - " + "name: " + name;
        }

    }
}
```

Скриншот:



Прочитайте TableView онлайн: https://riptutorial.com/ru/javafx/topic/2229/tableview

глава 6: WebView и WebEngine

замечания

`WebView` - это узел JavaFX, который интегрирован в дерево компонентов JavaFX. Он управляет `WebEngine` и отображает его содержимое.

`WebEngine` является основным `WebEngine` , который в основном выполняет всю работу.

Examples

Загрузка страницы

```
WebView wv = new WebView();
WebEngine we = wv.getEngine();
we.load("https://stackoverflow.com");
```

`WebView` - это оболочка пользовательского интерфейса вокруг `WebEngine` . Почти все элементы управления для взаимодействия с пользователем не связаны с классом `WebEngine` .

Получить историю страниц WebView

```
WebHistory history = webView.getEngine().getHistory();
```

История - это в основном список записей. Каждая запись представляет посещенную страницу и обеспечивает доступ к соответствующей информации о странице, такой как URL, название и дата последнего посещения страницы.

Список можно получить, используя метод `getEntries()` . История и соответствующий список записей изменяются, так как `WebEngine` перемещается по сети. Список может расширяться или сокращаться в зависимости от действий браузера. Эти изменения можно прослушать с помощью API ObservableList, который раскрывает список.

Индекс записи истории, связанный с текущей посещенной страницей, представлен `currentIndexProperty()` . Текущий индекс можно использовать для перехода к любой записи в истории с помощью метода `go(int)` . `maxSizeProperty()` устанавливает максимальный размер истории, который представляет собой размер списка истории

Ниже приведен пример получения и обработки списка элементов истории веб-поиска .

Для хранения элементов истории используется `ComboBox` (comboBox). Используя `ListChangeListener` в `WebHistory` `ComboBox` обновляется до текущей `WebHistory` . В `ComboBox` есть `EventHandler`

который перенаправляется на выбранную страницу.

```
final WebHistory history = webEngine.getHistory();

comboBox.setItems(history.getEntries());
comboBox.setPrefWidth(60);
comboBox.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent ev) {
        int offset =
                comboBox.getSelectionModel().getSelectedIndex()
                - history.getCurrentIndex();
        history.go(offset);
    }
});

history.currentIndexProperty().addListener(new ChangeListener<Number>() {

    @Override
    public void changed(ObservableValue<? extends Number> observable, Number oldValue, Number
newValue) {
        // update currently selected combobox item
        comboBox.getSelectionModel().select(newValue.intValue());
    }
});

// set converter for value shown in the combobox:
//   display the urls
comboBox.setConverter(new StringConverter<WebHistory.Entry>() {

    @Override
    public String toString(WebHistory.Entry object) {
        return object == null ? null : object.getUrl();
    }

    @Override
    public WebHistory.Entry fromString(String string) {
        throw new UnsupportedOperationException();
    }
});
```

отправьте предупреждения Javascript с отображаемой веб-страницы в журнал приложений Java.

```
private final Logger logger = Logger.getLogger(getClass().getCanonicalName());

WebView webView = new WebView();
webEngine = webView.getEngine();

webEngine.setOnAlert(event -> logger.warning(() -> "JS alert: " + event.getData()));
```

Связь между Java-приложением и Javascript на веб-странице

При использовании WebView для отображения собственной пользовательской веб-страницы, и эта веб-страница содержит Javascript, возможно, потребуется установить

двустороннюю связь между программой Java и Javascript на веб-странице.

В этом примере показано, как настроить такое сообщение.

На веб-странице должно отображаться поле ввода и кнопка. При нажатии кнопки значение из поля ввода отправляется в приложение Java, которое обрабатывает его. После обработки результат отправляется на Javascript, который, в свою очередь, отображает результат на веб-странице.

Основной принцип заключается в том, что для связи с Javascript в Java объект создается в Java, который установлен на веб-странице. А для другого направления объект создается в Javascript и извлекается с веб-страницы.

Следующий код показывает часть Java, я сохранил все это в одном файле:

```
package com.sothawo.test;

import javafx.application.Application;
import javafx.concurrent.Worker;
import javafx.scene.Scene;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;
import netscape.javascript.JSObject;

import java.io.File;
import java.net.URL;

/**
 * @author P.J. Meisch (pj.meisch@sothawo.com).
 */
public class WebViewApplication extends Application {

    /** for communication to the Javascript engine. */
    private JSObject javascriptConnector;

    /** for communication from the Javascript engine. */
    private JavaConnector javaConnector = new JavaConnector();;

    @Override
    public void start(Stage primaryStage) throws Exception {
        URL url = new File("./js-sample.html").toURI().toURL();

        WebView webView = new WebView();
        final WebEngine webEngine = webView.getEngine();

        // set up the listener
        webEngine.getLoadWorker().stateProperty().addListener((observable, oldValue, newValue)
-> {
            if (Worker.State.SUCCEEDED == newValue) {
                // set an interface object named 'javaConnector' in the web engine's page
                JSObject window = (JSObject) webEngine.executeScript("window");
                window.setMember("javaConnector", javaConnector);

                // get the Javascript connector object.
                javascriptConnector = (JSObject) webEngine.executeScript("getJsConnector()");
            }
```

```
        });

        Scene scene = new Scene(webView, 300, 150);
        primaryStage.setScene(scene);
        primaryStage.show();

        // now load the page
        webEngine.load(url.toString());
    }

    public class JavaConnector {
        /**
         * called when the JS side wants a String to be converted.
         *
         * @param value
         *          the String to convert
         */
        public void toLowerCase(String value) {
            if (null != value) {
                javascriptConnector.call("showResult", value.toLowerCase());
            }
        }
    }
}
```

Когда страница загружается, объект `JavaConnector` (который определяется внутренним классом и создается как поле) устанавливается на веб-страницу этими вызовами:

```
JSObject window = (JSObject) webEngine.executeScript("window");
window.setMember("javaConnector", javaConnector);
```

Объект `javascriptConnector` извлекается с веб-страницы с помощью

```
javascriptConnector = (JSObject) webEngine.executeScript("getJsConnector()");
```

Когда `toLowerCase(String)` метод `toLowerCase(String)` из `JavaConnector` , переданное значение преобразуется и затем отправляется обратно через объект `javascriptConnector` .

И это код html и javascript:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Sample</title>
    </head>
    <body>
        <main>

            <div><input id="input" type="text"></div>
            <button onclick="sendToJava();">to lower case</button>
            <div id="result"></div>

        </main>

        <script type="text/javascript">
```

```
            function sendToJava () {
                var s = document.getElementById('input').value;
                javaConnector.toLowerCase(s);
            };

            var jsConnector = {
                showResult: function (result) {
                    document.getElementById('result').innerHTML = result;
                }
            };

            function getJsConnector() {
                return jsConnector;
            };
        </script>
    </body>
</html>
```

Функция `sendToJava` вызывает метод `JavaConnector` который был задан кодом Java:

```
function sendToJava () {
    var s = document.getElementById('input').value;
    javaConnector.toLowerCase(s);
};
```

и функция, вызываемая кодом Java для извлечения `javascriptConnector` возвращает объект `jsConnector` :

```
var jsConnector = {
    showResult: function (result) {
        document.getElementById('result').innerHTML = result;
    }
};

function getJsConnector() {
    return jsConnector;
};
```

Тип аргумента вызовов между Java и Javascript не ограничивается строками. Более подробная информация о возможных типах и конверсиях содержится в документе JSObject API .

Прочитайте WebView и WebEngine онлайн: https://riptutorial.com/ru/javafx/topic/5156/webview-и-webengine

глава 7: Windows

Examples

Создание нового окна

Чтобы показать некоторый контент в новом окне, необходимо создать `Stage` . После создания и инициализации `show` или `showAndWait` необходимо вызвать в объекте `Stage` :

```
// create sample content
Rectangle rect = new Rectangle(100, 100, 200, 300);
Pane root = new Pane(rect);
root.setPrefSize(500, 500);

Parent content = root;

// create scene containing the content
Scene scene = new Scene(content);

Stage window = new Stage();
window.setScene(scene);

// make window visible
window.show();
```

Примечание. Этот код необходимо выполнить в потоке приложений JavaFX.

Создание пользовательского диалога

Вы можете создавать настраиваемые диалоги, которые содержат много компонентов и выполняют множество функций. Он ведет себя как второй этап на стадии владельца. В следующем примере приложение, которое отображает человека в представлении таблицы основного этапа и создает человека в диалоговом окне (AddingPersonDialog), подготовлено. GUI, созданные SceneBuilder, но они могут быть созданы чистыми java-кодами.

Пример применения:

AppMain.java

```
package customdialog;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class AppMain extends Application {
```

```
    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("AppMain.fxml"));
        Scene scene = new Scene(root, 500, 500);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

AppMainController.java

```
package customdialog;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Modality;
import javafx.stage.Stage;

public class AppMainController implements Initializable {

    @FXML
    private TableView<Person> tvData;
    @FXML
    private TableColumn colId;
    @FXML
    private TableColumn colName;
    @FXML
    private TableColumn colAge;

    private ObservableList<Person> tvObservableList = FXCollections.observableArrayList();

    @FXML
    void onOpenDialog(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("AddPersonDialog.fxml"));
        Parent parent = fxmlLoader.load();
        AddPersonDialogController dialogController =
fxmlLoader.<AddPersonDialogController>getController();
        dialogController.setAppMainObservableList(tvObservableList);

        Scene scene = new Scene(parent, 300, 200);
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setScene(scene);
        stage.showAndWait();
```

```
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));
        colAge.setCellValueFactory(new PropertyValueFactory<>("age"));
        tvData.setItems(tvObservableList);
    }

}
```

AppMain.fxml

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>

<AnchorPane maxHeight="400.0" minHeight="400.0" minWidth="500.0"
xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="customdialog.AppMainController">
   <children>
      <VBox alignment="CENTER" layoutX="91.0" layoutY="85.0" spacing="10.0"
AnchorPane.bottomAnchor="30.0" AnchorPane.leftAnchor="30.0" AnchorPane.rightAnchor="30.0"
AnchorPane.topAnchor="30.0">
         <children>
            <Button mnemonicParsing="false" onAction="#onOpenDialog" text="Add Person" />
            <TableView fx:id="tvData" prefHeight="300.0" prefWidth="400.0">
              <columns>
                <TableColumn fx:id="colId" prefWidth="75.0" text="ID" />
                <TableColumn fx:id="colName" prefWidth="75.0" text="Name" />
                  <TableColumn fx:id="colAge" prefWidth="75.0" text="Age" />
              </columns>
               <columnResizePolicy>
                  <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
               </columnResizePolicy>
            </TableView>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

AddPersonDialogController.java

```
package customdialog;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class AddPersonDialogController  {
```

```
    @FXML
    private TextField tfId;

    @FXML
    private TextField tfName;

    @FXML
    private TextField tfAge;

    private ObservableList<Person> appMainObservableList;

    @FXML
    void btnAddPersonClicked(ActionEvent event) {
        System.out.println("btnAddPersonClicked");
        int id = Integer.valueOf(tfId.getText().trim());
        String name = tfName.getText().trim();
        int iAge = Integer.valueOf(tfAge.getText().trim());

        Person data = new Person(id, name, iAge);
        appMainObservableList.add(data);

        closeStage(event);
    }

    public void setAppMainObservableList(ObservableList<Person> tvObservableList) {
        this.appMainObservableList = tvObservableList;

    }

    private void closeStage(ActionEvent event) {
        Node  source = (Node)  event.getSource();
        Stage stage  = (Stage) source.getScene().getWindow();
        stage.close();
    }

}
```

AddPersonDialog.fxml

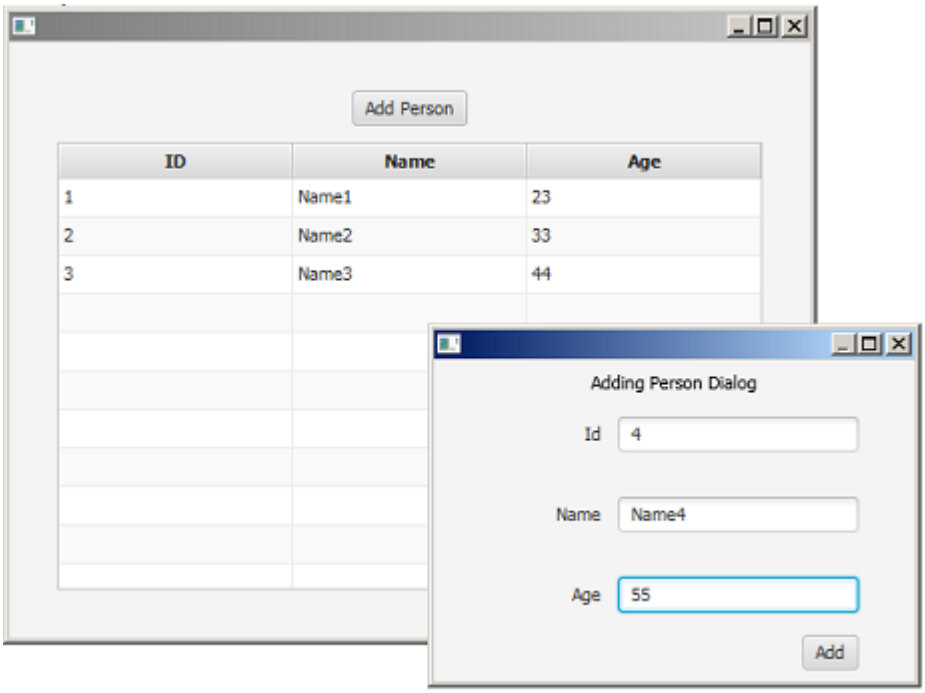
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>

<AnchorPane minHeight="300.0" minWidth="400.0" xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="customdialog.AddPersonDialogController">
   <children>
      <VBox alignment="CENTER" layoutX="131.0" layoutY="50.0" prefHeight="200.0"
prefWidth="100.0" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="5.0"
AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="5.0">
         <children>
            <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Adding Person Dialog" />
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
```

```
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Id" />
                  <TextField fx:id="tfId" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Name" />
                  <TextField fx:id="tfName" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Age" />
                  <TextField fx:id="tfAge" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER_RIGHT">
               <children>
                  <Button mnemonicParsing="false" onAction="#btnAddPersonClicked" text="Add"
/>
               </children>
               <opaqueInsets>
                  <Insets />
               </opaqueInsets>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

Person.java

```
package customdialog;

import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;

public class Person {

    private SimpleIntegerProperty id;
    private SimpleStringProperty name;
    private SimpleIntegerProperty age;

    public Person(int id, String name, int age)  {
        this.id = new SimpleIntegerProperty(id);
        this.name = new SimpleStringProperty(name);
        this.age = new SimpleIntegerProperty(age);
    }
```

```
    public int getId() {
        return id.get();
    }

    public void setId(int ID) {
        this.id.set(ID);
    }

    public String getName() {
        return name.get();
    }

    public void setName(String nme) {
        this.name.set(nme);
    }

    public int getAge() {
        return age.get();
    }

    public void setAge(int age) {
        this.age.set(age);
    }

    @Override
    public String toString() {
        return "id: " + id.get() + " - " + "name: " + name.get()+ "age: "+ age.get();
    }

}
```



Скриншот

Создание пользовательского диалога

Вы можете создавать настраиваемые диалоги, которые содержат много компонентов и выполняют множество функций. Он ведет себя как второй этап на стадии владельца.

В следующем примере приложение, которое отображает человека в представлении таблицы основного этапа и создает человека в диалоговом окне (AddingPersonDialog), подготовлено. GUI, созданные SceneBuilder, но они могут быть созданы чистыми java-кодами.

Пример применения:

AppMain.java

```
package customdialog;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class AppMain extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("AppMain.fxml"));
        Scene scene = new Scene(root, 500, 500);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

AppMainController.java

```
package customdialog;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Modality;
import javafx.stage.Stage;

public class AppMainController implements Initializable {

    @FXML
    private TableView<Person> tvData;
    @FXML
    private TableColumn colId;
```

```
    @FXML
    private TableColumn colName;
    @FXML
    private TableColumn colAge;

    private ObservableList<Person> tvObservableList = FXCollections.observableArrayList();

    @FXML
    void onOpenDialog(ActionEvent event) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("AddPersonDialog.fxml"));
        Parent parent = fxmlLoader.load();
        AddPersonDialogController dialogController =
fxmlLoader.<AddPersonDialogController>getController();
        dialogController.setAppMainObservableList(tvObservableList);

        Scene scene = new Scene(parent, 300, 200);
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setScene(scene);
        stage.showAndWait();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        colId.setCellValueFactory(new PropertyValueFactory<>("id"));
        colName.setCellValueFactory(new PropertyValueFactory<>("name"));
        colAge.setCellValueFactory(new PropertyValueFactory<>("age"));
        tvData.setItems(tvObservableList);
    }

}
```

AppMain.fxml

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>

<AnchorPane maxHeight="400.0" minHeight="400.0" minWidth="500.0"
xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="customdialog.AppMainController">
   <children>
      <VBox alignment="CENTER" layoutX="91.0" layoutY="85.0" spacing="10.0"
AnchorPane.bottomAnchor="30.0" AnchorPane.leftAnchor="30.0" AnchorPane.rightAnchor="30.0"
AnchorPane.topAnchor="30.0">
         <children>
            <Button mnemonicParsing="false" onAction="#onOpenDialog" text="Add Person" />
            <TableView fx:id="tvData" prefHeight="300.0" prefWidth="400.0">
              <columns>
                <TableColumn fx:id="colId" prefWidth="75.0" text="ID" />
                <TableColumn fx:id="colName" prefWidth="75.0" text="Name" />
                  <TableColumn fx:id="colAge" prefWidth="75.0" text="Age" />
              </columns>
               <columnResizePolicy>
                  <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
```

```
               </columnResizePolicy>
            </TableView>
         </children>
      </VBox>
   </children>
</AnchorPane>
```

AddPersonDialogController.java

```
package customdialog;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class AddPersonDialogController  {

    @FXML
    private TextField tfId;

    @FXML
    private TextField tfName;

    @FXML
    private TextField tfAge;

    private ObservableList<Person> appMainObservableList;

    @FXML
    void btnAddPersonClicked(ActionEvent event) {
        System.out.println("btnAddPersonClicked");
        int id = Integer.valueOf(tfId.getText().trim());
        String name = tfName.getText().trim();
        int iAge = Integer.valueOf(tfAge.getText().trim());

        Person data = new Person(id, name, iAge);
        appMainObservableList.add(data);

        closeStage(event);
    }

    public void setAppMainObservableList(ObservableList<Person> tvObservableList) {
        this.appMainObservableList = tvObservableList;

    }

    private void closeStage(ActionEvent event) {
        Node  source = (Node)  event.getSource();
        Stage stage  = (Stage) source.getScene().getWindow();
        stage.close();
    }

}
```

AddPersonDialog.fxml

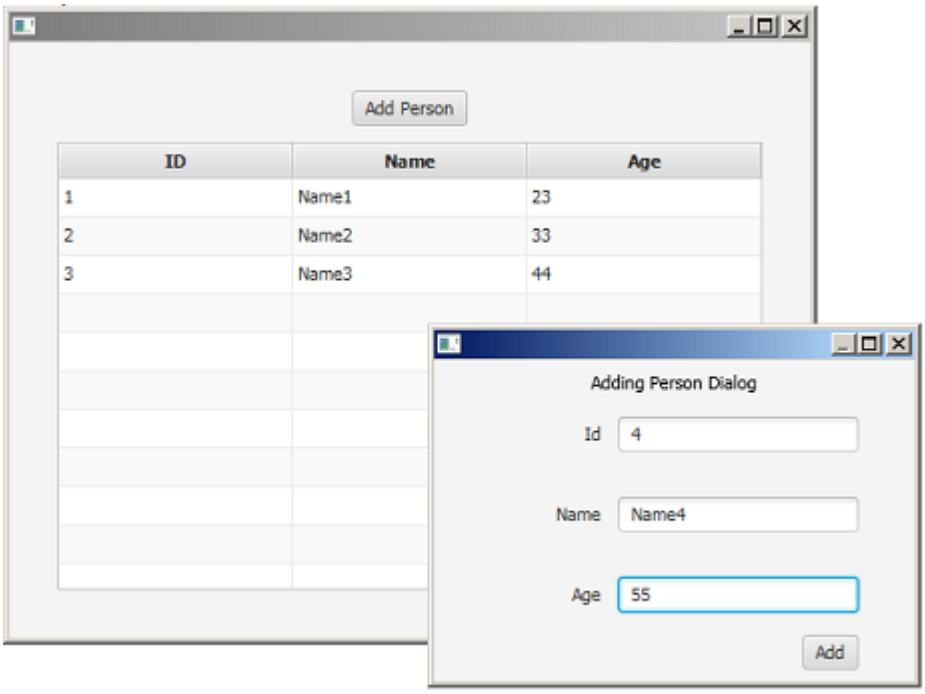
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>

<AnchorPane minHeight="300.0" minWidth="400.0" xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="customdialog.AddPersonDialogController">
   <children>
      <VBox alignment="CENTER" layoutX="131.0" layoutY="50.0" prefHeight="200.0"
prefWidth="100.0" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="5.0"
AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="5.0">
         <children>
            <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Adding Person Dialog" />
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Id" />
                  <TextField fx:id="tfId" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Name" />
                  <TextField fx:id="tfName" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER" prefHeight="50.0" prefWidth="200.0" spacing="10.0">
               <children>
                  <Label alignment="CENTER_RIGHT" minWidth="100.0" text="Age" />
                  <TextField fx:id="tfAge" HBox.hgrow="ALWAYS" />
               </children>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
            <HBox alignment="CENTER_RIGHT">
               <children>
                  <Button mnemonicParsing="false" onAction="#btnAddPersonClicked" text="Add"
/>
               </children>
               <opaqueInsets>
                  <Insets />
               </opaqueInsets>
               <padding>
                  <Insets right="30.0" />
               </padding>
            </HBox>
         </children>
      </VBox>
   </children>
```

```
</AnchorPane>
```

Person.java

```
package customdialog;

import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;

public class Person {

    private SimpleIntegerProperty id;
    private SimpleStringProperty name;
    private SimpleIntegerProperty age;

    public Person(int id, String name, int age)  {
        this.id = new SimpleIntegerProperty(id);
        this.name = new SimpleStringProperty(name);
        this.age = new SimpleIntegerProperty(age);
    }

    public int getId() {
        return id.get();
    }

    public void setId(int ID) {
        this.id.set(ID);
    }

    public String getName() {
        return name.get();
    }

    public void setName(String nme) {
        this.name.set(nme);
    }

    public int getAge() {
        return age.get();
    }

    public void setAge(int age) {
        this.age.set(age);
    }

    @Override
    public String toString() {
        return "id: " + id.get() + " - " + "name: " + name.get()+ "age: "+ age.get();
    }

}
```

Скриншот



Прочитайте Windows онлайн: https://riptutorial.com/ru/javafx/topic/1496/windows

глава 8: Анимация

Examples

Анимация объекта с временной шкалой

```
Button button = new Button("I'm here...");

Timeline t = new Timeline(
        new KeyFrame(Duration.seconds(0), new KeyValue(button.translateXProperty(), 0)),
        new KeyFrame(Duration.seconds(2), new KeyValue(button.translateXProperty(), 80))
);
t.setAutoReverse(true);
t.setCycleCount(Timeline.INDEFINITE);
t.play();
```

Самый простой и гибкий способ использования анимации в JavaFX - это класс `Timeline` . Временная шкала работает с использованием `KeyFrame` s как известных точек анимации. В этом случае он знает, что в начале (`0 seconds`), что `translateXProperty` должен быть равен нулю, а в конце (`2 seconds`), что свойство должно быть `80` . Вы также можете делать другие вещи, например, настроить анимацию на обратное и сколько раз она должна выполняться.

Сроки могут анимировать несколько свойств одновременно:

```
Timeline t = new Timeline(
        new KeyFrame(Duration.seconds(0), new KeyValue(button.translateXProperty(), 0)),
        new KeyFrame(Duration.seconds(1), new KeyValue(button.translateYProperty(), 10)),
        new KeyFrame(Duration.seconds(2), new KeyValue(button.translateXProperty(), 80)),
        new KeyFrame(Duration.seconds(3), new KeyValue(button.translateYProperty(), 90))
);                                                              //   ^ notice X vs Y
```

Эта анимация примет свойство `Y` от `0` (начальное значение свойства) до `10` секунд в секунду, и оно заканчивается на `90` секунд в три секунды. Обратите внимание, что когда анимация начинается с того, что `Y` возвращается к нулю, даже если это не первое значение на временной шкале.

Прочитайте Анимация онлайн: https://riptutorial.com/ru/javafx/topic/5166/анимация

глава 9: Диаграмма

Examples

Круговая диаграмма

Класс `PieChart` рисует данные в виде круга, который разделен на срезы. Каждый срез представляет процент (часть) для определенного значения. Данные круговой диаграммы обернуты в объекты `PieChart.Data` . Каждый объект `PieChart.Data` имеет два поля: имя среза пирога и соответствующее ему значение.

Конструкторы

Чтобы создать круговую диаграмму, нам нужно создать объект класса `PieChart` . Мы даем два конструктора. Один из них создает пустую диаграмму, которая ничего не отображает, если данные не заданы с помощью метода `setData` :

```
PieChart pieChart = new PieChart(); // Creates an empty pie chart
```

А вторая требует, чтобы в качестве параметра был передан список `ObservableList` данных `PieChart.Data ObservableList` .

```
ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Cats", 50),
            new PieChart.Data("Dogs", 50));
PieChart pieChart(valueList); // Creates a chart with the given data
```

Данные

Значения кусочков пирога необязательно должны суммироваться до 100, так как размер среза будет рассчитываться пропорционально сумме всех значений.

Порядок, в котором записи данных будут добавлены в список, определит их позицию на графике. По умолчанию они отложены по часовой стрелке, но это поведение можно изменить:

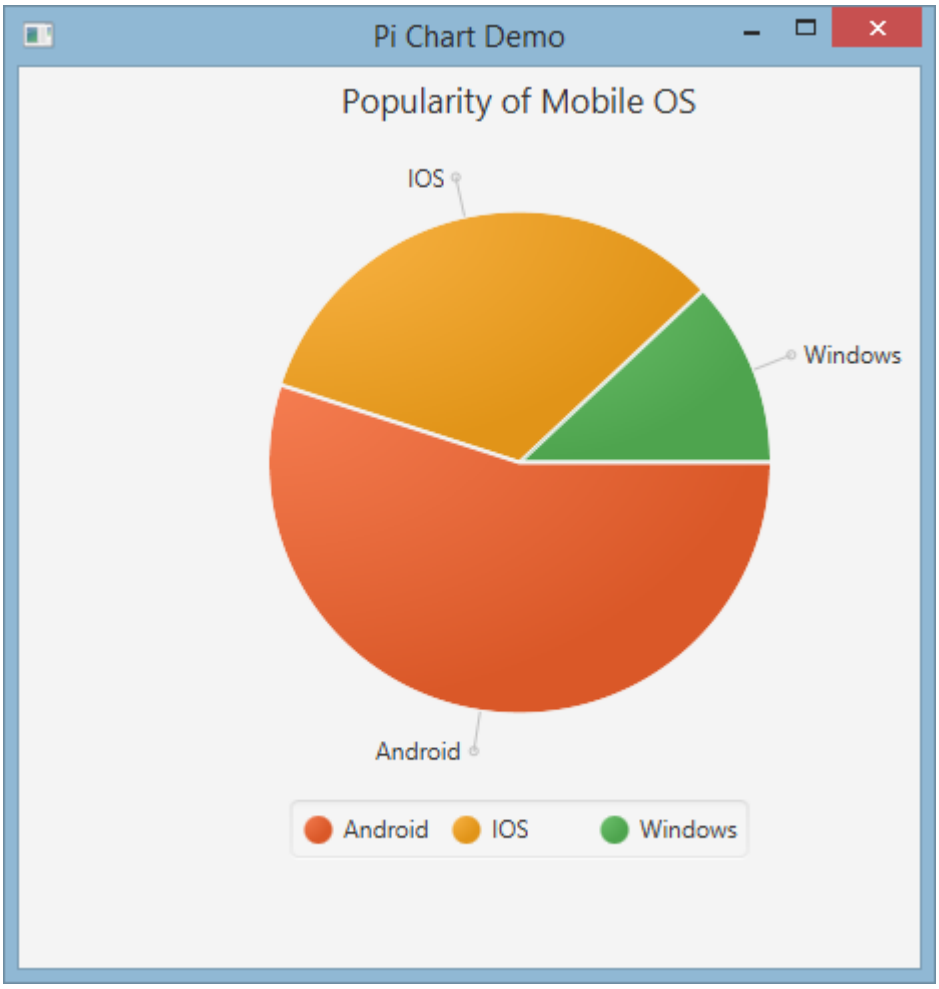
```
pieChart.setClockwise(false);
```

пример

В следующем примере создается простая круговая диаграмма:

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.chart.PieChart;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;

public class Main extends Application {

@Override
public void start(Stage primaryStage) {
    Pane root = new Pane();
    ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Android", 55),
            new PieChart.Data("IOS", 33),
            new PieChart.Data("Windows", 12));
    // create a pieChart with valueList data.
    PieChart pieChart = new PieChart(valueList);
    pieChart.setTitle("Popularity of Mobile OS");
    //adding pieChart to the root.
    root.getChildren().addAll(pieChart);
    Scene scene = new Scene(root, 450, 450);

    primaryStage.setTitle("Pie Chart Demo");
    primaryStage.setScene(scene);
    primaryStage.show();
}

public static void main(String[] args) {
        launch(args);
    }
}
```

Выход:



Интерактивная круговая диаграмма

По умолчанию `PieChart` не обрабатывает никаких событий, но это поведение можно изменить, потому что каждый кусочек пирога является `Node` JavaFX.

В приведенном ниже примере мы инициализируем данные, присваиваем их диаграмме, а затем перебираем набор данных, добавляя всплывающие подсказки к каждому фрагменту, так что значения, обычно скрытые, могут быть представлены пользователю.

```
ObservableList<PieChart.Data> valueList = FXCollections.observableArrayList(
            new PieChart.Data("Nitrogen", 7809),
            new PieChart.Data("Oxygen", 2195),
            new PieChart.Data("Other", 93));

PieChart pieChart = new PieChart(valueList);
pieChart.setTitle("Air composition");

pieChart.getData().forEach(data -> {
    String percentage = String.format("%.2f%%", (data.getPieValue() / 100));
    Tooltip toolTip = new Tooltip(percentage);
    Tooltip.install(data.getNode(), toolTip);
});
```

Линейный график

Класс `LineChart` представляет данные как ряд точек данных, связанных с прямыми линиями. Каждая точка данных завернута в объект `XYChart.Data` , а точки данных сгруппированы в `XYChart.Series` .

Каждый объект `XYChart.Data` имеет два поля, к которым можно получить доступ, используя `getXValue` и `getYValue` , которые соответствуют значению x и ay на диаграмме.

```
XYChart.Data data = new XYChart.Data(1,3);
System.out.println(data.getXValue()); // Will print 1
System.out.println(data.getYValue()); // Will print 3
```

Топоры

Прежде чем мы создадим `LineChart` нам нужно определить его оси. Например, конструктор no-argument по `NumberAxis` класса `NumberAxis` создаст ось автоматического `NumberAxis` которая готова к использованию и не нуждается в дальнейшей настройке.

```
Axis xAxis = new NumberAxis();
```

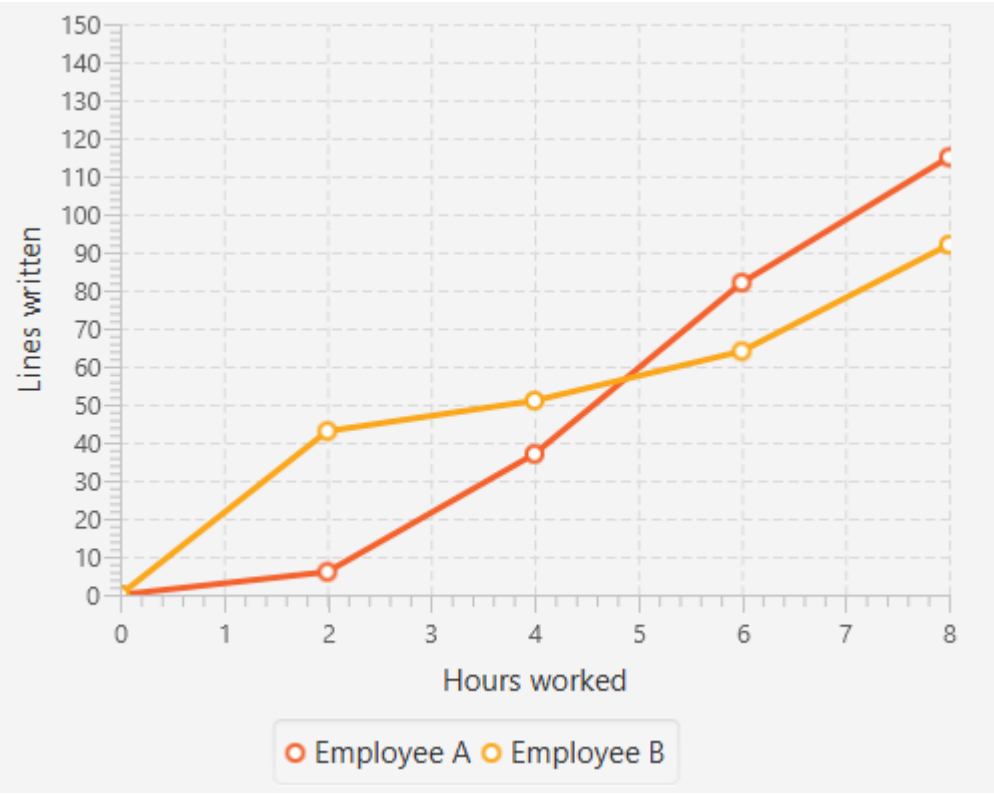
пример

В приведенном ниже примере мы создаем две серии данных, которые будут отображаться на одной диаграмме. Ярлыки, диапазоны и значения меток обозначены явно.

```
@Override
public void start(Stage primaryStage) {
    Pane root = new Pane();

    // Create empty series
    ObservableList<XYChart.Series> seriesList = FXCollections.observableArrayList();

    // Create data set for the first employee and add it to the series
    ObservableList<XYChart.Data> aList = FXCollections.observableArrayList(
            new XYChart.Data(0, 0),
            new XYChart.Data(2, 6),
            new XYChart.Data(4, 37),
            new XYChart.Data(6, 82),
            new XYChart.Data(8, 115)
    );
    seriesList.add(new XYChart.Series("Employee A", aList));

    // Create data set for the second employee and add it to the series
    ObservableList<XYChart.Data> bList = FXCollections.observableArrayList(
            new XYChart.Data(0, 0),
            new XYChart.Data(2, 43),
            new XYChart.Data(4, 51),
```

```
            new XYChart.Data(6, 64),
            new XYChart.Data(8, 92)
    );
    seriesList.add(new XYChart.Series("Employee B", bList));

    // Create axes
    Axis xAxis = new NumberAxis("Hours worked", 0, 8, 1);
    Axis yAxis = new NumberAxis("Lines written", 0, 150, 10);

    LineChart chart = new LineChart(xAxis, yAxis, seriesList);

    root.getChildren().add(chart);

    Scene scene = new Scene(root);
    primaryStage.setScene(scene);
    primaryStage.show();
}

public static void main(String[] args) {
    launch(args);
}
```

Выход:



Прочитайте Диаграмма онлайн: https://riptutorial.com/ru/javafx/topic/2631/диаграмма

глава 10: Диалоги

замечания

Диалоги были добавлены в обновление JavaFX 8 40.

Examples

TextInputDialog

`TextInputDialog` позволяет попросить пользователя ввести одну `String` .

```
TextInputDialog dialog = new TextInputDialog("42");
dialog.setHeaderText("Input your favourite int.");
dialog.setTitle("Favourite number?");
dialog.setContentText("Your favourite int: ");

Optional<String> result = dialog.showAndWait();

String s = result.map(r -> {
    try {
        Integer n = Integer.valueOf(r);
        return MessageFormat.format("Nice! I like {0} too!", n);
    } catch (NumberFormatException ex) {
        return MessageFormat.format("Unfortunately \"{0}\" is not a int!", r);
    }
}).orElse("You really don't want to tell me, huh?");

System.out.println(s);
```

ChoiceDialog

`ChoiceDialog` позволяет пользователю выбрать один элемент из списка параметров.

```
List<String> options = new ArrayList<>();
options.add("42");
options.add("9");
options.add("467829");
options.add("Other");

ChoiceDialog<String> dialog = new ChoiceDialog<>(options.get(0), options);
dialog.setHeaderText("Choose your favourite number.");
dialog.setTitle("Favourite number?");
dialog.setContentText("Your favourite number:");

Optional<String> choice = dialog.showAndWait();

String s = choice.map(c -> "Other".equals(c) ?
              "Unfortunately your favourite number is not available!"
              : "Nice! I like " + c + " too!")
           .orElse("You really don't want to tell me, huh?");
```

```
System.out.println(s);
```

бдительный

`Alert` - это простое всплывающее окно, которое отображает набор кнопок и получает результат в зависимости от кнопки, которую пользователь нажал:

пример

Это позволяет пользователю решить, если он действительно хочет закрыть основной этап:

```
@Override
public void start(Stage primaryStage) {
    Scene scene = new Scene(new Group(), 100, 100);

    primaryStage.setOnCloseRequest(evt -> {
        // allow user to decide between yes and no
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION, "Do you really want to close
this application?", ButtonType.YES, ButtonType.NO);

        // clicking X also means no
        ButtonType result = alert.showAndWait().orElse(ButtonType.NO);

        if (ButtonType.NO.equals(result)) {
            // consume event i.e. ignore close request
            evt.consume();
        }
    });
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Обратите внимание , что текст кнопки автоматически регулируется в зависимости от `Locale` .

Пользовательский текст кнопки

Текст, отображаемый кнопкой, можно настроить, создав экземпляр `ButtonType` самостоятельно:

```
ButtonType answer = new ButtonType("42");
ButtonType somethingElse = new ButtonType("54");

Alert alert = new Alert(Alert.AlertType.NONE, "What do you get when you multiply six by
nine?", answer, somethingElse);
ButtonType result = alert.showAndWait().orElse(somethingElse);

Alert resultDialog = new Alert(Alert.AlertType.INFORMATION,
                               answer.equals(result) ? "Correct" : "wrong",
                               ButtonType.OK);

resultDialog.show();
```

Прочитайте Диалоги онлайн: https://riptutorial.com/ru/javafx/topic/3681/диалоги

глава 11: Интернационализация в JavaFX

Examples

Загрузка пакета ресурсов

JavaFX обеспечивает простой способ интернационализировать ваши пользовательские интерфейсы. При создании представления из файла FXML вы можете предоставить `FXMLLoader` пакет ресурсов:

```
Locale locale = new Locale("en", "UK");
ResourceBundle bundle = ResourceBundle.getBundle("strings", locale);

Parent root = FXMLLoader.load(getClass().getClassLoader()
                                    .getResource("ui/main.fxml"), bundle);
```

Этот предоставленный комплект автоматически используется для перевода всех текстов в ваш файл FXML, начинающийся с `%` . `strings_en_UK.properties` ваш файл свойств `strings_en_UK.properties` содержит следующую строку:

```
ui.button.text=I'm a Button
```

Если в вашем FXML есть определение кнопки:

```
<Button text="%ui.button.text"/>
```

Он автоматически получит перевод для ключа `ui.button.text` .

контроллер

Пакеты ресурсов содержат локальные объекты. Вы можете передать пакет на `FXMLLoader` во время его создания. Контроллер должен реализовать интерфейс `Initializable` и переопределить метод `initialize(URL location, ResourceBundle resources)` . Второй параметр этого метода - `ResourceBundle` который передается из FXMLLoader в контроллер и может использоваться контроллером для дальнейшего перевода текстов или изменения другой информации, зависящей от локали.

```
public class MyController implements Initializable {

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        label.setText(resources.getString("country"));
    }
}
```

Динамическое переключение языка при запуске приложения

В этом примере показано, как создать приложение JavaFX, где язык может переключаться динамически во время работы приложения.

Это файлы пакета сообщений, используемые в примере:

messages_en.properties :

```
window.title=Dynamic language change
button.english=English
button.german=German
label.numSwitches=Number of language switches: {0}
```

messages_de.properties :

```
window.title=Dynamischer Sprachwechsel
button.english=Englisch
button.german=Deutsch
label.numSwitches=Anzahl Sprachwechsel: {0}
```

Основная идея состоит в том, чтобы иметь класс утилиты I18N (в качестве альтернативы это может быть реализовано singleton).

```
import javafx.beans.binding.Bindings;
import javafx.beans.binding.StringBinding;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Locale;
import java.util.ResourceBundle;
import java.util.concurrent.Callable;

/**
 * I18N utility class..
 */
public final class I18N {

    /** the current selected Locale. */
    private static final ObjectProperty<Locale> locale;

    static {
        locale = new SimpleObjectProperty<>(getDefaultLocale());
        locale.addListener((observable, oldValue, newValue) -> Locale.setDefault(newValue));
    }

    /**
     * get the supported Locales.
     *
```

```
     * @return List of Locale objects.
     */
    public static List<Locale> getSupportedLocales() {
        return new ArrayList<>(Arrays.asList(Locale.ENGLISH, Locale.GERMAN));
    }

    /**
     * get the default locale. This is the systems default if contained in the supported
locales, english otherwise.
     *
     * @return
     */
    public static Locale getDefaultLocale() {
        Locale sysDefault = Locale.getDefault();
        return getSupportedLocales().contains(sysDefault) ? sysDefault : Locale.ENGLISH;
    }

    public static Locale getLocale() {
        return locale.get();
    }

    public static void setLocale(Locale locale) {
        localeProperty().set(locale);
        Locale.setDefault(locale);
    }

    public static ObjectProperty<Locale> localeProperty() {
        return locale;
    }

    /**
     * gets the string with the given key from the resource bundle for the current locale and
uses it as first argument
     * to MessageFormat.format, passing in the optional args and returning the result.
     *
     * @param key
     *         message key
     * @param args
     *         optional arguments for the message
     * @return localized formatted string
     */
    public static String get(final String key, final Object... args) {
        ResourceBundle bundle = ResourceBundle.getBundle("messages", getLocale());
        return MessageFormat.format(bundle.getString(key), args);
    }

    /**
     * creates a String binding to a localized String for the given message bundle key
     *
     * @param key
     *         key
     * @return String binding
     */
    public static StringBinding createStringBinding(final String key, Object... args) {
        return Bindings.createStringBinding(() -> get(key, args), locale);
    }

    /**
     * creates a String Binding to a localized String that is computed by calling the given
func
     *
```

```
     * @param func
     *          function called on every change
     * @return StringBinding
     */
    public static StringBinding createStringBinding(Callable<String> func) {
        return Bindings.createStringBinding(func, locale);
    }

    /**
     * creates a bound Label whose value is computed on language change.
     *
     * @param func
     *          the function to compute the value
     * @return Label
     */
    public static Label labelForValue(Callable<String> func) {
        Label label = new Label();
        label.textProperty().bind(createStringBinding(func));
        return label;
    }

    /**
     * creates a bound Button for the given resourcebundle key
     *
     * @param key
     *          ResourceBundle key
     * @param args
     *          optional arguments for the message
     * @return Button
     */
    public static Button buttonForKey(final String key, final Object... args) {
        Button button = new Button();
        button.textProperty().bind(createStringBinding(key, args));
        return button;
    }
}
```

Этот класс имеет статическое поле `locale` , которая является Java `Locale` объекта , завернутым в JavaFX `ObjectProperty` , так что переплеты могут быть созданы для этого свойства. Первые методы - это стандартные методы получения и установки свойства JavaFX.

`get(final String key, final Object... args)` - это основной метод, который используется для реального извлечения сообщения из `ResourceBundle` .

Два метода с именем `createStringBinding` создают `StringBinding` , привязанный к `locale` и поэтому привязки будут меняться всякий раз, когда изменяется свойство `locale` . Первый использует свои аргументы для извлечения и форматирования сообщения с использованием метода `get` упомянутого выше, второй передается в `Callable` , который должен вызывать новое строковое значение.

Последние два метода - это методы создания компонентов JavaFX. Первый метод используется для создания `Label` и использует `Callable` для его внутренней привязки строк. Второй создает `Button` и использует значение ключа для извлечения привязки String.

Конечно, можно было бы создать много других объектов, таких как `MenuItem` или `ToolTip` но этих двух должно быть достаточно для примера.

Этот код показывает, как этот класс используется в приложении:

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

import java.util.Locale;

/**
 * Sample application showing dynamic language switching,
 */
public class I18nApplication extends Application {

    /** number of language switches. */
    private Integer numSwitches = 0;

    @Override
    public void start(Stage primaryStage) throws Exception {

        primaryStage.titleProperty().bind(I18N.createStringBinding("window.title"));

        // create content
        BorderPane content = new BorderPane();

        // at the top two buttons
        HBox hbox = new HBox();
        hbox.setPadding(new Insets(5, 5, 5, 5));
        hbox.setSpacing(5);

        Button buttonEnglish = I18N.buttonForKey("button.english");
        buttonEnglish.setOnAction((evt) -> switchLanguage(Locale.ENGLISH));
        hbox.getChildren().add(buttonEnglish);

        Button buttonGerman = I18N.buttonForKey("button.german");
        buttonGerman.setOnAction((evt) -> switchLanguage(Locale.GERMAN));
        hbox.getChildren().add(buttonGerman);

        content.setTop(hbox);

        // a label to display the number of changes, recalculating the text on every change
        final Label label = I18N.labelForValue(() -> I18N.get("label.numSwitches",
numSwitches));
        content.setBottom(label);

        primaryStage.setScene(new Scene(content, 400, 200));
        primaryStage.show();
    }

    /**
     * sets the given Locale in the I18N class and keeps count of the number of switches.
     *
     * @param locale
```

```
     *          the new local to set
     */
    private void switchLanguage(Locale locale) {
        numSwitches++;
        I18N.setLocale(locale);
    }
}
```

Приложение показывает три разных способа использования `StringBinding` созданных классом `I18N` :

1. заголовок окна связан напрямую с помощью `StringBinding` .
2. кнопки используют вспомогательный метод с помощью клавиш сообщения
3. метка использует вспомогательный метод с помощью `Callable` . Этот `Callable` использует метод `I18N.get()` для получения форматированной переведенной строки, содержащей фактическое количество переключателей.

При нажатии кнопки счетчик увеличивается, и `I18N` свойство locale `I18N` s, которое, в свою очередь, вызывает изменение привязок строк и, таким образом, устанавливает строку пользовательского интерфейса в новые значения.

Прочитайте Интернационализация в JavaFX онлайн:
https://riptutorial.com/ru/javafx/topic/5434/интернационализация-в-javafx

глава 12: кнопка

Examples

Добавление прослушивателя действий

Кнопки запускают события срабатывания при их активации (например, нажатие, привязка клавиш для кнопки нажата, ...).

```
button.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});
```

Если вы используете Java 8+, вы можете использовать lambdas для прослушивателей действий.

```
button.setOnAction((ActionEvent a) -> System.out.println("Hello, World!"));
// or
button.setOnAction(a -> System.out.println("Hello, World!"));
```

Добавление графика к кнопке

Кнопки могут иметь графику. `graphic` может быть любой узел JavaFX, например `ProgressBar`

```
button.setGraphic(new ProgressBar(-1));
```

`ImageView`

```
button.setGraphic(new ImageView("images/icon.png"));
```

Или еще одна кнопка

```
button.setGraphic(new Button("Nested button"));
```

Создать кнопку

Создание `Button` прост:

```
Button sampleButton = new Button();
```

Это создаст новую `Button` без текста или рисунка внутри.

Если вы хотите создать `Button` с текстом, просто используйте конструктор, который принимает параметр `String` as (который устанавливает `textProperty` `Button`):

```
Button sampleButton = new Button("Click Me!");
```

Если вы хотите создать `Button` с графикой внутри или любым другим `Node` , используйте этот конструктор:

```
Button sampleButton = new Button("I have an icon", new ImageView(new Image("icon.png")));
```

Кнопки по умолчанию и Отмена

`Button` API предоставляет простой способ назначить общие сочетания клавиш кнопкам без необходимости доступа к списку ускорителей, назначенных для `Scene` или явно прослушиванию ключевых событий. А именно, предоставляются два удобных метода: `setDefaultButton` и `setCancelButton` :

- Установка `setDefaultButton` в значение `true` приведет к срабатыванию `Button` при каждом событии `KeyCode.ENTER` .
- Установка `setCancelButton` в значение `true` приведет к срабатыванию `Button` каждый раз, когда он получает событие `KeyCode.ESCAPE` .

В следующем примере создается `Scene` с двумя кнопками, которые запускаются при нажатии клавиш ввода или выхода, независимо от того, сфокусированы они или нет.

```
FlowPane root = new FlowPane();

Button okButton = new Button("OK");
okButton.setDefaultButton(true);
okButton.setOnAction(e -> {
    System.out.println("OK clicked.");
});

Button cancelButton = new Button("Cancel");
cancelButton.setCancelButton(true);
cancelButton.setOnAction(e -> {
    System.out.println("Cancel clicked.");
});

root.getChildren().addAll(okButton, cancelButton);
Scene scene = new Scene(root);
```

Приведенный выше код не будет работать, если эти `KeyEvents` будут использованы любым родительским `Node` :

```
scene.setOnKeyPressed(e -> {
    e.consume();
});
```

Прочитайте кнопка онлайн: https://riptutorial.com/ru/javafx/topic/5162/кнопка

глава 13: Макеты

Examples

StackPane

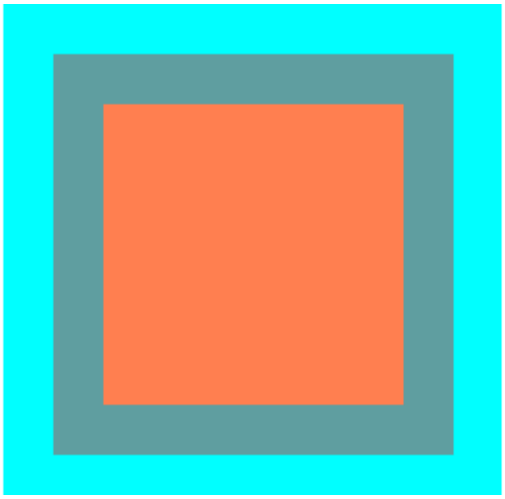
`StackPane` своих детей в стеке «назад к фронту».

Z-порядок детей определяется порядком списка детей (доступным по вызову `getChildren`): 0-й ребенок является нижним и последним дочерним элементом поверх стека.

Стекловая панель пытается изменить размер каждого дочернего элемента, чтобы заполнить свою собственную область содержимого. В случае, если дочерний объект не может быть изменен, чтобы заполнить область `StackPane` (либо потому, что он не был изменен, либо его максимальный размер предотвратил его), тогда он будет выровнен по области, используя `alignmentProperty` стека, по умолчанию - `Pos.CENTER`.

пример

```
// Create a StackPane
StackPane pane = new StackPane();

// Create three squares
Rectangle rectBottom = new Rectangle(250, 250);
rectBottom.setFill(Color.AQUA);
Rectangle rectMiddle = new Rectangle(200, 200);
rectMiddle.setFill(Color.CADETBLUE);
Rectangle rectUpper = new Rectangle(150, 150);
rectUpper.setFill(Color.CORAL);

// Place them on top of each other
pane.getChildren().addAll(rectBottom, rectMiddle, rectUpper);
```



HBox и VBox

`HBox` и `VBox` очень похожи, оба выкладывают своих детей в одну строку.

Общие характеристики

Если у `HBox` или `VBox` есть граница и / или набор дополнений, тогда содержимое будет выложено внутри этих вложений.

Они выставляют каждого управляемого ребенка независимо от значения видимого свойства ребенка; неуправляемые дети игнорируются.

Выравнивание содержимого контролируется свойством выравнивания, которое по умолчанию **`Pos.TOP_LEFT`** .

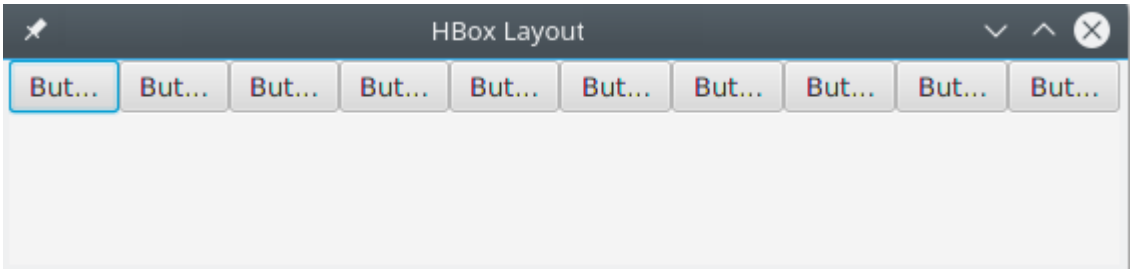
HBox

`HBox` своих детей в одном горизонтальном ряду слева направо.

`HBox` будет изменять размер дочерних `HBox` (если они изменяются) **до их предпочтительной ширины** s и использует свойство fillHeight, чтобы определить, изменять ли их высоты, чтобы заполнить свою собственную высоту или сохранить их высоты до их предпочтительного значения (по умолчанию значения fillHeight равны true).

Создание HBox

```
// HBox example
HBox row = new HBox();
Label first = new Label("First");
Label second = new Label("Second");
row.getChildren().addAll(first, second);
```



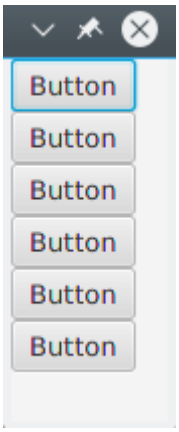
VBox

`VBox` выставляет своих детей в одной вертикальной колонке сверху вниз.

`VBox` будет изменять размеры дочерних элементов (если они изменяются) **до их предпочтительных высот** и использует свойство fillWidth, чтобы определить, изменять ли их ширину, чтобы заполнить свою собственную ширину или сохранить их ширину до их предпочтительных значений (по умолчанию значение fillWidth равно true).

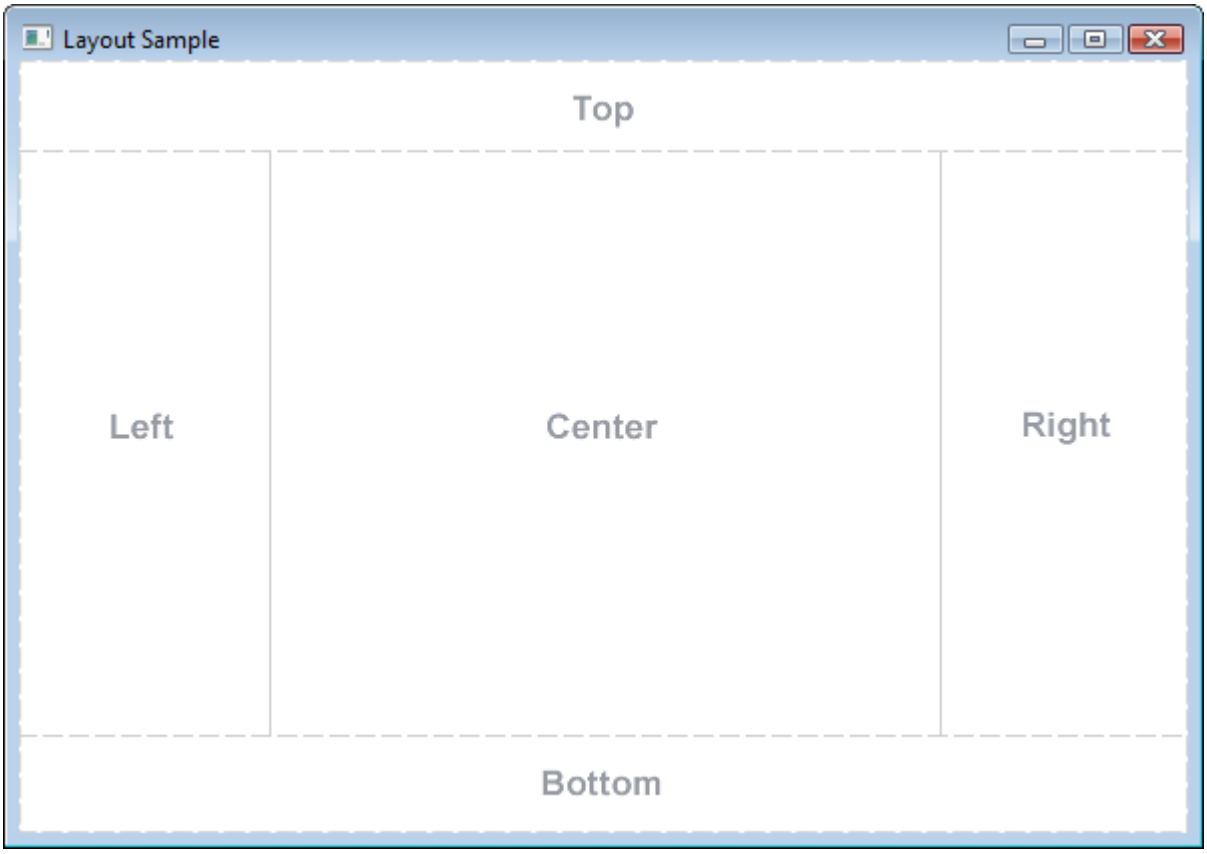
Создание VBox

```
// VBox example
VBox column = new VBox();
Label upper = new Label("Upper");
Label lower = new Label("Lower");
column.getChildren().addAll(upper, lower);
```



BorderPane

`BorderPane` разделен на пять разных областей.



Приграничные области (`Top` , `Right` , `Bottom` , `Left`) имеют предпочтительный размер в зависимости от их содержимого. По умолчанию они будут использовать только то, что им нужно, в то время как область `Center` займет любое оставшееся пространство. Когда пограничные области пусты, они не занимают места.

Каждая область может содержать только один элемент. Его можно добавить с помощью

методов `setTop(Node)` , `setRight(Node)` , `setBottom(Node)` , `setLeft(Node)` , `setCenter(Node)` . Вы можете использовать другие макеты, чтобы поместить более одного элемента в одну область.

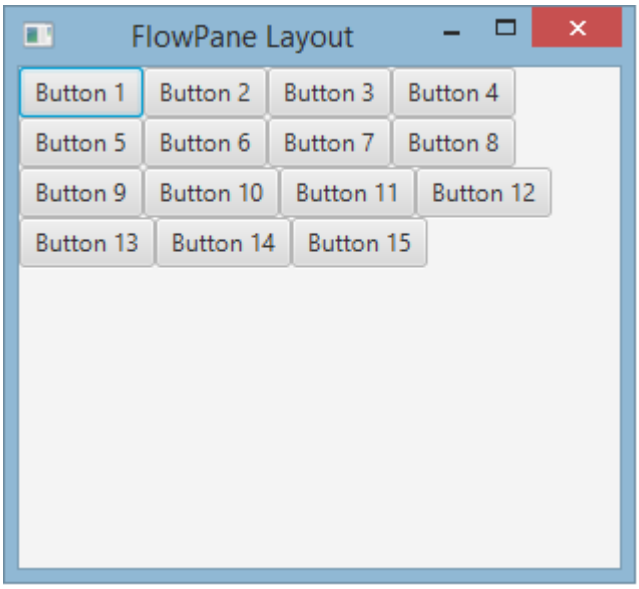
```
//BorderPane example
BorderPane pane = new BorderPane();

Label top = new Label("Top");

Label right = new Label("Right");

HBox bottom = new HBox();
bottom.getChildren().addAll(new Label("First"), new Label("Second"));

VBox left = new VBox();
left.getChildren().addAll(new Label("Upper"), new Label("Lower"));

StackPane center = new StackPane();
center.getChildren().addAll(new Label("Lorem"), new Label("ipsum"));

pane.setTop(top);        //The text "Top"
pane.setRight(right);    //The text "Right"
pane.setBottom(bottom);  //Row of two texts
pane.setLeft(left);      //Column of two texts
pane.setCenter(center);  //Two texts on each other
```

FlowPane

`FlowPane` устанавливает узлы в строках или столбцах на основе доступных доступных горизонтальных или вертикальных пространств. Он переносит узлы на следующую строку, когда горизонтальное пространство меньше, чем общая ширина всех узлов; он переносит узлы в следующий столбец, когда вертикальное пространство меньше, чем общее количество всех высот узлов. Этот пример иллюстрирует горизонтальную компоновку по умолчанию:

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception{
        FlowPane root = new FlowPane();
        for (int i=1; i<=15; i++) {
            Button b1=new Button("Button "+String.valueOf(i));
            root.getChildren().add(b1); //for adding button to root
        }
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("FlowPane Layout");
        primaryStage.setScene(scene);
        primaryStage.show();
```

```
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



`FlowPane` умолчанию:

```
FlowPane root = new FlowPane();
```

Дополнительные конструкторы `FlowPane` :

```
FlowPane() //Creates a horizontal FlowPane layout with hgap/vgap = 0 by default.
FlowPane(double hgap, double gap) //Creates a horizontal FlowPane layout with the specified
hgap/vgap.
FlowPane(double hgap, double vgap, Node... children) //Creates a horizontal FlowPane layout
with the specified hgap/vgap.
FlowPane(Node... children) //Creates a horizontal FlowPane layout with hgap/vgap = 0.
FlowPane(Orientation orientation) //Creates a FlowPane layout with the specified orientation
and hgap/vgap = 0.
FlowPane(Orientation orientation, double hgap, double gap) //Creates a FlowPane layout with
the specified orientation and hgap/vgap.
FlowPane(Orientation orientation, double hgap, double vgap, Node... children) //Creates a
FlowPane layout with the specified orientation and hgap/vgap.
FlowPane(Orientation orientation, Node... children) //Creates a FlowPane layout with the
specified orientation and hgap/vgap = 0.
```

Добавление узлов в макет использует методы `add()` или `addAll()` родительской `Pane` :

```
Button btn = new Button("Demo Button");
root.getChildren().add(btn);
root.getChildren().addAll(…);
```

По умолчанию `FlowPane` дочерние узлы слева направо. Чтобы изменить выравнивание потока, вызовите метод `setAlignment()` , передав перечислимое значение типа `Pos` .

Некоторые обычно используемые выравнивания потока:

```
root.setAlignment(Pos.TOP_RIGHT);     //for top right
root.setAlignment(Pos.TOP_CENTER);    //for top Center
root.setAlignment(Pos.CENTER);        //for Center
root.setAlignment(Pos.BOTTOM_RIGHT);  //for bottom right
```

GridPane

`GridPane` своих дочерних `GridPane` в гибкой сетке строк и столбцов.

Дети грид-панели

Ребенок может быть размещен в любом месте `GridPane` и может охватывать несколько строк / столбцов (по умолчанию - 1), а его размещение в сетке определяется ограничениями макета:

скованность	Описание
ColumnIndex	столбец, где начинается область макета ребенка.
RowIndex	строка, где начинается область макета ребенка.
ColumnSpan	количество столбцов, на которые распространяется макет макета, горизонтально.
RowSpan	количество строк, в которых располагается область макета ребенка, вертикально.

Общее количество строк / столбцов не нужно указывать заранее, так как gridpane автоматически расширяет / сокращает сетку для размещения контента.

Добавление детей в GridPane

Чтобы добавить новый `Node` в `GridPane` **ограничения макета** для `GridPane` должны быть установлены с использованием статического метода класса `GridPane` , после чего эти дети могут быть добавлены в экземпляр `GridPane` .

```
GridPane gridPane = new GridPane();

// Set the constraints: first row and first column
Label label = new Label("Example");
GridPane.setRowIndex(label, 0);
GridPane.setColumnIndex(label, 0);
// Add the child to the grid
gridpane.getChildren().add(label);
```

`GridPane` предоставляет удобные методы для объединения этих шагов:

```
gridPane.add(new Button("Press me!"), 1, 0); // column=1 row=0
```

Класс `GridPane` также предоставляет статические методы настройки для установки дочерних элементов **row- и columnspan** :

```
Label labelLong = new Label("Its a long text that should span several rows");
GridPane.setColumnSpan(labelLong, 2);
gridPane.add(labelLong, 0, 1);  // column=0 row=1
```

Размер столбцов и строк

По умолчанию строки и столбцы будут иметь размер, соответствующий их контенту. В случае необходимости **явного контроля размеров** `RowConstraints` и `ColumnConstraints` можно добавить экземпляры `RowConstraints` и `GridPane` . Добавление этих двух ограничений приведет к изменению размера вышеприведенного примера, чтобы иметь первый столбец 100 пикселей, а второй столбец - 200 пикселей.

```
gridPane.getColumnConstraints().add(new ColumnConstraints(100));
gridPane.getColumnConstraints().add(new ColumnConstraints(200));
```

По умолчанию `GridPane` будет изменять размеры строк / столбцов до их предпочтительных размеров, даже если размер сетки будет больше размера, чем его предпочтительный размер. Для поддержки **динамических размеров столбцов** / **строк** оба класса contstaints предоставляют три свойства: минимальный размер, максимальный размер и предпочтительный размер.

Дополнительно `ColumnConstraints` предоставляет `setHGrow` и `RowConstraints` предоставляет методы `setVGrow` чтобы **повлиять на приоритет роста и сокращения** . Три заранее определенных приоритета:

- **Priority.ALWAYS** : Всегда старайтесь расти (или уменьшаться), разделяя увеличение (или уменьшение) пространства с другими областями макета, которые растут (или уменьшаются) ВСЕГДА
- **Priority.SOMETIMES** : Если нет других областей макета с ростом (или уменьшением), установленным ВСЕГДА, или эти области макета не поглощают все увеличенное (или уменьшенное) пространство, то будут разделять увеличение (или уменьшение) в пространстве с помощью другие области размещения SOMETIMES.
- **Priority.NEVER** : область макета никогда не будет расти (или уменьшаться), когда есть увеличение (или уменьшение) пространства, доступного в регионе.

```
ColumnConstraints column1 = new ColumnConstraints(100, 100, 300);
column1.setHgrow(Priority.ALWAYS);
```

Столбец, определенный выше, имеет минимальный размер 100 пикселей, и он всегда будет пытаться расти до тех пор, пока он не достигнет максимальной ширины 300 пикселей.

Также можно определить **процентное соотношение** для строк и столбцов. В следующем примере определяется `GridPane` где первый столбец заполняет 40% ширины сетки, второй заполняет 60%.

```
GridPane gridpane = new GridPane();
ColumnConstraints column1 = new ColumnConstraints();
column1.setPercentWidth(40);
ColumnConstraints column2 = new ColumnConstraints();
column2.setPercentWidth(60);
gridpane.getColumnConstraints().addAll(column1, column2);
```

Выравнивание элементов внутри ячеек сетки

Выравнивание `Node` с может быть определено с помощью `setHalignment` (по горизонтали) метод `ColumnConstraints` класса и `setValignment` (вертикальной) методом `RowConstraints` класса.

```
ColumnConstraints column1 = new ColumnConstraints();
column1.setHalignment(HPos.RIGHT);

RowConstraints row1 = new RowConstraints();
row1.setValignment(VPos.CENTER);
```

TilePane

Макет панели плитки похож на макет FlowPane. TilePane помещает все узлы в сетку, в которой каждая ячейка или плитка имеют одинаковый размер. Он упорядочивает узлы в аккуратных строках и столбцах, горизонтально или вертикально.

```
    import javafx.application.Application;
    import javafx.scene.Scene;
    import javafx.scene.control.Button;
    import javafx.scene.layout.TilePane;
    import javafx.stage.Stage;

    public class Main extends Application {

        @Override
        public void start(Stage primaryStage) {
            primaryStage.setTitle("TilePane Demo");
            double width = 400;
            double height = 300;
            TilePane root = new TilePane();
            root.setStyle("-fx-background-color:blue");
            // to set horizontal and vertical gap
            root.setHgap(20);
            root.setVgap(50);
            Button bl = new Button("Buttons");
            root.getChildren().add(bl);
            Button btn = new Button("Button");
```

```
            root.getChildren().add(btn);
            Button btn1 = new Button("Button 1");
            root.getChildren().add(btn1);
            Button btn2 = new Button("Button 2");
            root.getChildren().add(btn2);
            Button btn3 = new Button("Button 3");
            root.getChildren().add(btn3);
            Button btn4 = new Button("Button 4");
            root.getChildren().add(btn4);

            Scene scene = new Scene(root, width, height);
            primaryStage.setScene(scene);
            primaryStage.show();
        }


        public static void main(String[] args) {
            launch(args);
        }
    }
```

ВЫХОД

Чтобы создать Tilepane

```
TilePane root = new TilePane();
```

setHgap () И метод setVgap () используется для создания разрыва между столбцом и столбцом. мы также можем установить столбцы для макета, используя

```
int columnCount = 2;
root.setPrefColumns(columnCount);
```

AnchorPane

`AnchorPane` а - это макет, который позволяет размещать контент на определенном расстоянии от его сторон.

Существует 4 метода настройки и 4 метода для получения расстояний в `AnchorPane` . Первым параметром этих методов является дочерний `Node` . Второй параметр сеттеров - это `Double` значение для использования. Это значение может быть `null` что не указывает на ограничение для данной стороны.

метод сеттера	метод геттера
setBottomAnchor	getBottomAnchor
setLeftAnchor	getLeftAnchor
setRightAnchor	getRightAnchor
setTopAnchor	getTopAnchor

В следующем примере места размещаются на заданных расстояниях от сторон.

`center` региона также изменяется , чтобы сохранить заданные расстояния от сторон. Наблюдайте за поведением при изменении размера окна.

```
public static void setBackgroundColor(Region region, Color color) {
    // change to 50% opacity
    color = color.deriveColor(0, 1, 1, 0.5);
    region.setBackground(new Background(new BackgroundFill(color, CornerRadii.EMPTY,
Insets.EMPTY)));
}

@Override
public void start(Stage primaryStage) {
    Region right = new Region();
    Region top = new Region();
    Region left = new Region();
    Region bottom = new Region();
    Region center = new Region();

    right.setPrefSize(50, 150);
    top.setPrefSize(150, 50);
    left.setPrefSize(50, 150);
    bottom.setPrefSize(150, 50);

    // fill with different half-transparent colors
    setBackgroundColor(right, Color.RED);
    setBackgroundColor(left, Color.LIME);
    setBackgroundColor(top, Color.BLUE);
    setBackgroundColor(bottom, Color.YELLOW);
    setBackgroundColor(center, Color.BLACK);

    // set distances to sides
    AnchorPane.setBottomAnchor(bottom, 50d);
    AnchorPane.setTopAnchor(top, 50d);
    AnchorPane.setLeftAnchor(left, 50d);
```

```
    AnchorPane.setRightAnchor(right, 50d);

    AnchorPane.setBottomAnchor(center, 50d);
    AnchorPane.setTopAnchor(center, 50d);
    AnchorPane.setLeftAnchor(center, 50d);
    AnchorPane.setRightAnchor(center, 50d);

    // create AnchorPane with specified children
    AnchorPane anchorPane = new AnchorPane(left, top, right, bottom, center);

    Scene scene = new Scene(anchorPane, 200, 200);

    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Прочитайте Макеты онлайн: https://riptutorial.com/ru/javafx/topic/2121/макеты

глава 14: Многопоточность

Examples

Обновление пользовательского интерфейса с помощью Platform.runLater

Длительные операции не должны запускаться в потоке приложений JavaFX, так как это предотвращает обновление JavaFX пользовательского интерфейса, что приводит к замороженному интерфейсу.

Кроме того, любое изменение в `Node` которое является частью «живого» графика сцены, **должно** происходить в потоке приложений JavaFX. `Platform.runLater` можно использовать для выполнения этих обновлений в потоке приложений JavaFX.

В следующем примере показано, как обновлять `Text Node` несколько раз из другого потока:

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class CounterApp extends Application {

    private int count = 0;
    private final Text text = new Text(Integer.toString(count));

    private void incrementCount() {
        count++;
        text.setText(Integer.toString(count));
    }

    @Override
    public void start(Stage primaryStage) {
        StackPane root = new StackPane();
        root.getChildren().add(text);

        Scene scene = new Scene(root, 200, 200);

        // longrunning operation runs on different thread
        Thread thread = new Thread(new Runnable() {

            @Override
            public void run() {
                Runnable updater = new Runnable() {

                    @Override
                    public void run() {
                        incrementCount();
                    }
                };
```

```
                while (true) {
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException ex) {
                    }

                    // UI update is run on the Application thread
                    Platform.runLater(updater);
                }
            }

        });
        // don't let thread prevent JVM shutdown
        thread.setDaemon(true);
        thread.start();

        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

Группировка обновлений пользовательского интерфейса

Следующий код делает пользовательский интерфейс неактивным в течение короткого времени после нажатия кнопки, так как используется слишком много вызовов `Platform.runLater` . (Попробуйте прокрутить `ListView` сразу после нажатия кнопки.)

```
@Override
public void start(Stage primaryStage) {
    ObservableList<Integer> data = FXCollections.observableArrayList();
    ListView<Integer> listView = new ListView<>(data);

    Button btn = new Button("Say 'Hello World'");
    btn.setOnAction((ActionEvent event) -> {
        new Thread(() -> {
            for (int i = 0; i < 100000; i++) {
                final int index = i;
                Platform.runLater(() -> data.add(index));
            }
        }).start();
    });

    Scene scene = new Scene(new VBox(listView, btn));

    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Чтобы предотвратить это, вместо использования большого количества обновлений, следующий код использует `AnimationTimer` для запуска обновления только один раз для каждого кадра:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.AnimationTimer;

public class Updater {

    @FunctionalInterface
    public static interface UpdateTask {

        public void update() throws Exception;
    }

    private final List<UpdateTask> updates = new ArrayList<>();

    private final AnimationTimer timer = new AnimationTimer() {

        @Override
        public void handle(long now) {
            synchronized (updates) {
                for (UpdateTask r : updates) {
                    try {
                        r.update();
                    } catch (Exception ex) {
                        Logger.getLogger(Updater.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
                updates.clear();
                stop();
            }
        }
    };

    public void addTask(UpdateTask... tasks) {
        synchronized (updates) {
            updates.addAll(Arrays.asList(tasks));
            timer.start();
        }
    }

}
```

который позволяет группировать обновления с использованием класса `Updater` :

```
private final Updater updater = new Updater();

...

        // Platform.runLater(() -> data.add(index));
        updater.addTask(() -> data.add(index));
```

Как использовать службу JavaFX

Вместо того, чтобы запускать интенсивные задачи в `JavaFX Thread` который должен быть сделан в `Service` что же такое служба ?

Служба - это класс, который создает новый `Thread` каждый раз, когда вы его запускаете, и передаёт ему задачу, чтобы выполнить некоторую работу. Служба может вернуть или не значение.

Ниже приведен типичный пример службы JavaFX, которая выполняет некоторую работу и возвращает `Map<String,String>()`:

```
public class WorkerService extends Service<Map<String, String>> {

    /**
     * Constructor
     */
    public WorkerService () {

        // if succeeded
        setOnSucceeded(s -> {
            //code if Service succeeds
        });

        // if failed
        setOnFailed(fail -> {
            //code it Service fails
        });

        //if cancelled
        setOnCancelled(cancelled->{
            //code if Service get's cancelled
        });
    }

    /**
    * This method starts the Service
    */
    public void startTheService(){
        if(!isRunning()){
           //...
           reset();
           start();
        }

    }

    @Override
    protected Task<Map<String, String>> createTask() {
        return new Task<Map<String, String>>() {
            @Override
            protected Void call() throws Exception {

                    //create a Map<String, String>
                    Map<String,String> map  = new HashMap<>();

                   //create other variables here

                   try{
                        //some code here
                        //.....do your manipulation here

                        updateProgress(++currentProgress, totalProgress);
                    }
```

```
                } catch (Exception ex) {
                    return null; //something bad happened so you have to do something instead
of returning null
                }

                return map;
            }
        };
    }

}
```

Прочитайте Многопоточность онлайн: https://riptutorial.com/ru/javafx/topic/2230/многопоточность

глава 15: пагинация

Examples

Создание разбивки на страницы

Паганины в JavaFX используют обратный вызов для получения страниц, используемых в анимации.

```
Pagination p = new Pagination();
p.setPageFactory(param -> new Button(param.toString()));
```

Это создает бесконечный список кнопок с нумерацией `0..` так как нулевой конструктор arg создает бесконечную разбивку на страницы. `setPageFactory` принимает обратный вызов, который принимает int, и возвращает узел, который мы хотим в этом индексе.

Автоматическое продвижение

```
Pagination p = new Pagination(10);

Timeline fiveSecondsWonder = new Timeline(new KeyFrame(Duration.seconds(5), event -> {
    int pos = (p.getCurrentPageIndex()+1) % p.getPageCount();
    p.setCurrentPageIndex(pos);
}));
fiveSecondsWonder.setCycleCount(Timeline.INDEFINITE);
fiveSecondsWonder.play();

stage.setScene(new Scene(p));
stage.show();
```

Это увеличивает паузы каждые 5 секунд.

Как это устроено

```
Pagination p = new Pagination(10);

Timeline fiveSecondsWonder = new Timeline(new KeyFrame(Duration.seconds(5), event -> {
```

`fiveSecondsWonder` - это временная шкала, которая запускает событие каждый раз, когда заканчивается цикл. В этом случае время цикла составляет 5 секунд.

```
    int pos = (p.getCurrentPageIndex()+1) % p.getPageCount();
    p.setCurrentPageIndex(pos);
```

Отметьте разбивку на страницы.

```
}));
fiveSecondsWonder.setCycleCount(Timeline.INDEFINITE);
```

Установите временную шкалу для запуска навсегда.

```
fiveSecondsWonder.play();
```

Создание разбиения на страницы изображений

```
ArrayList<String> images = new ArrayList<>();
images.add("some\\cool\\image");
images.add("some\\other\\cool\\image");
images.add("some\\cooler\\image");

Pagination p = new Pagination(3);
p.setPageFactory(n -> new ImageView(images.get(n)));
```

Обратите внимание, что пути должны быть URL-адресами, а не файловыми путями.

Как это устроено

```
p.setPageFactory(n -> new ImageView(images.get(n)));
```

Все остальное - просто пух, вот где происходит настоящая работа. `setPageFactory` принимает обратный вызов, который принимает int, и возвращает узел, который мы хотим в этом индексе. Первая страница отображает первый элемент в списке, второй - второй элемент в списке и т. Д.

Прочитайте пагинация онлайн: https://riptutorial.com/ru/javafx/topic/5165/пагинация

глава 16: Переключатель

Examples

Создание кнопок радиосвязи

Кнопки «Радио» позволяют вам выбрать один элемент из указанных. Существует два способа объявить `RadioButton` с текстом, кроме него. Либо используя конструктор по умолчанию `RadioButton()` установив текст с помощью метода `setText(String)` или используя другой конструктор `RadioButton(String)` .

```
RadioButton radioButton1 = new RadioButton();
radioButton1.setText("Select me!");
RadioButton radioButton2= new RadioButton("Or me!");
```

Поскольку `RadioButton` является расширением `Labeled` также может быть `Image` указанное в `RadioButton` . После создания `RadioButton` с одним из конструкторов просто добавьте `Image` с помощью `setGraphic(ImageView)` как здесь:

```
Image image = new Image("ok.jpg");
RadioButton radioButton = new RadioButton("Agree");
radioButton.setGraphic(new ImageView(image));
```

Использовать группы на радио-кнопках

`ToggleGroup` используется для управления `RadioButton` s, так что каждый раз в каждой группе может быть выбран один.

Создайте простую `ToggleGroup` как `ToggleGroup` ниже:

```
ToggleGroup group = new ToggleGroup();
```

После создания `Togglegroup` его можно назначить на `RadioButton` s, используя `setToggleGroup(ToggleGroup)` . Используйте `setSelected(Boolean)` для предварительного выбора одного из `RadioButton` .

```
RadioButton radioButton1 = new RadioButton("stackoverlow is awesome! :)");
radioButton1.setToggleGroup(group);
radioButton1.setSelected(true);

RadioButton radioButton2 = new RadioButton("stackoverflow is ok :|");
radioButton2.setToggleGroup(group);

RadioButton radioButton3 = new RadioButton("stackoverflow is useless :(");
radioButton3.setToggleGroup(group);
```

События для радиоустройств

Как правило, когда один из `RadioButton` s в `ToggleGroup` выбран, приложение выполняет действие. Ниже приведен пример, который печатает пользовательские данные выбранного `RadioButton` который был установлен с помощью `setUserData(Object)` .

```
radioButton1.setUserData("awesome")
radioButton2.setUserData("ok");
radioButton3.setUserData("useless");

ToggleGroup group = new ToggleGroup();
group.selectedToggleProperty().addListener((obserableValue, old_toggle, new_toggle) -> {
    if (group.getSelectedToggle() != null) {
        System.out.println("You think that stackoverflow is " +
group.getSelectedToggle().getUserData().toString());
    }
});
```

Запрос фокуса для кнопок радио

Предположим, что второй `RadioButton` из трех предварительно выбран с помощью `setSelected(Boolean)` , фокус по-прежнему находится на первом `RadioButton` по умолчанию. Чтобы изменить это, используйте метод `requestFocus()` .

```
radioButton2.setSelected(true);
radioButton2.requestFocus();
```

Прочитайте Переключатель онлайн: https://riptutorial.com/ru/javafx/topic/5906/переключатель

глава 17: печать

Examples

Основная печать

```
PrinterJob pJ = PrinterJob.createPrinterJob();

if (pJ != null) {
    boolean success = pJ.printPage(some-node);
    if (success) {
        pJ.endJob();
    }
}
```

Это печатает на принтере по умолчанию, не показывая никакого диалога пользователю. Чтобы использовать принтер, отличный от стандартного, вы можете использовать `PrinterJob#createPrinterJob(Printer)` для установки текущего принтера. Вы можете использовать это, чтобы просмотреть все принтеры в своей системе:

```
System.out.println(Printer.getAllPrinters());
```

Печать с системным диалогом

```
PrinterJob pJ = PrinterJob.createPrinterJob();

if (pJ != null) {
    boolean success = pJ.showPrintDialog(primaryStage);// this is the important line
    if (success) {
        pJ.endJob();
    }
}
```

Прочитайте печать онлайн: https://riptutorial.com/ru/javafx/topic/5157/печать

глава 18: Свойства и наблюдаемые

замечания

Свойства можно наблюдать, и к ним можно добавить слушателей. Они последовательно используются для свойств `Node` .

Examples

Типы свойств и наименований

Стандартные свойства

В зависимости от типа свойства существует до 3 методов для одного свойства. Пусть `<property>` обозначает имя свойства и `<Property>` имя свойства с первой буквой в верхнем регистре. И пусть `T` - тип свойства; для примитивных оболочек мы используем здесь примитивный тип, например `String` для `StringProperty` и `double` для `ReadOnlyDoubleProperty` .

Имя метода	параметры	Тип возврата	Цель
`<property>Property`	`()`	Само свойство, например `DoubleProperty` , `ReadOnlyStringProperty` , `ObjectProperty<VPos>`	вернуть свойство для добавления слушателей / привязки
`get<Property>`	`()`	`T`	вернуть значение, завернутое в свойство
`set<Property>`	`(T)`	`void`	установить значение свойства

Обратите внимание, что сеттер не существует для свойств readonly.

Свойства списка Readonly

Свойства Readonly list - это свойства, которые предоставляют только метод getter. Тип такого свойства - `ObservableList` , предпочтительно с указанным типом agrument. Значение этого свойства никогда не изменяется; вместо этого может быть изменено содержимое `ObservableList` .

Свойства Readonly map

Подобно свойствам readonly list свойства readonly map предоставляют только getter, и

содержимое может быть изменено вместо значения свойства. Геттер возвращает `ObservableMap` .

Пример StringProperty

В следующем примере показано объявление свойства (`StringProperty` в этом случае) и демонстрируется, как добавить к нему `ChangeListener` .

```
import java.text.MessageFormat;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;

public class Person {

    private final StringProperty name = new SimpleStringProperty();

    public final String getName() {
        return this.name.get();
    }

    public final void setName(String value) {
        this.name.set(value);
    }

    public final StringProperty nameProperty() {
        return this.name;
    }

    public static void main(String[] args) {
        Person person = new Person();
        person.nameProperty().addListener(new ChangeListener<String>() {

            @Override
            public void changed(ObservableValue<? extends String> observable, String oldValue,
String newValue) {
                System.out.println(MessageFormat.format("The name changed from \"{0}\" to
\"{1}\"", oldValue, newValue));
            }

        });

        person.setName("Anakin Skywalker");
        person.setName("Darth Vader");
    }

}
```

Пример ReadOnlyIntegerProperty

В этом примере показано, как использовать свойство оболочки readonly для создания свойства, которое невозможно записать. В этом случае `cost` и `price` могут быть изменены, но `profit` всегда будет `price - cost` .

```
import java.text.MessageFormat;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ReadOnlyIntegerProperty;
import javafx.beans.property.ReadOnlyIntegerWrapper;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;

public class Product {

    private final IntegerProperty price = new SimpleIntegerProperty();
    private final IntegerProperty cost = new SimpleIntegerProperty();
    private final ReadOnlyIntegerWrapper profit = new ReadOnlyIntegerWrapper();

    public Product() {
        // the property itself can be written to
        profit.bind(price.subtract(cost));
    }

    public final int getCost() {
        return this.cost.get();
    }

    public final void setCost(int value) {
        this.cost.set(value);
    }

    public final IntegerProperty costProperty() {
        return this.cost;
    }

    public final int getPrice() {
        return this.price.get();
    }

    public final void setPrice(int value) {
        this.price.set(value);
    }

    public final IntegerProperty priceProperty() {
        return this.price;
    }

    public final int getProfit() {
        return this.profit.get();
    }

    public final ReadOnlyIntegerProperty profitProperty() {
        // return a readonly view of the property
        return this.profit.getReadOnlyProperty();
    }

    public static void main(String[] args) {
        Product product = new Product();
        product.profitProperty().addListener(new ChangeListener<Number>() {

            @Override
            public void changed(ObservableValue<? extends Number> observable, Number oldValue,
Number newValue) {
                System.out.println(MessageFormat.format("The profit changed from {0}$ to
{1}$", oldValue, newValue));
```

```
            }

        });
        product.setCost(40);
        product.setPrice(50);
        product.setCost(20);
        product.setPrice(30);
    }

}
```

Прочитайте Свойства и наблюдаемые онлайн: https://riptutorial.com/ru/javafx/topic/4436/свойства-и-наблюдаемые

глава 19: Связывание JavaFX

Examples

Простое связывание свойств

JavaFX имеет API привязки, который предоставляет способы привязки одного свойства к другому. Это означает, что всякий раз, когда значение одного свойства изменяется, значение связанного свойства обновляется автоматически. Пример простой привязки:

```
SimpleIntegerProperty first =new SimpleIntegerProperty(5); //create a property with value=5
SimpleIntegerProperty second=new SimpleIntegerProperty();

public void test()
{
    System.out.println(second.get()); // '0'
    second.bind(first);               //bind second property to first
    System.out.println(second.get()); // '5'
    first.set(16);                    //set first property's value
    System.out.println(second.get()); // '16' - the value was automatically updated
}
```

Вы также можете связывать примитивное свойство с применением сложения, вычитания, деления и т. Д .:

```
public void test2()
{
        second.bind(first.add(100));
        System.out.println(second.get()); //'105'
        second.bind(first.subtract(50));
        System.out.println(second.get()); //'-45'
}
```

Любой объект может быть помещен в SimpleObjectProperty:

```
SimpleObjectProperty<Color> color=new SimpleObjectProperty<>(Color.web("45f3d1"));
```

Можно создать двунаправленные привязки. В этом случае свойства зависят друг от друга.

```
public void test3()
{
        second.bindBidirectional(first);
        System.out.println(second.get()+" "+first.get());
        second.set(1000);
        System.out.println(second.get()+" "+first.get()); //both are '1000'
}
```

Прочитайте Связывание JavaFX онлайн: https://riptutorial.com/ru/javafx/topic/7014/связывание-javafx

глава 20: Сценарий

Вступление

JavaFX Scene Builder - это инструмент визуального макета, который позволяет пользователям быстро создавать пользовательские интерфейсы приложений JavaFX без кодирования. Он используется для генерации файлов FXML.

замечания

JavaFX Scene Builder - это инструмент визуального макета, который позволяет пользователям быстро создавать пользовательские интерфейсы приложений JavaFX без кодирования. Пользователи могут перетаскивать компоненты пользовательского интерфейса в рабочую область, изменять их свойства, применять таблицы стилей, а код FXML для макета, который они создают, автоматически генерируется в фоновом режиме. Результатом является файл FXML, который затем может быть объединен с проектом Java, привязывая интерфейс к логике приложения.

С точки зрения контроллера модели (MVC):

- Файл FXML, содержащий описание пользовательского интерфейса, представляет собой представление.
- Контроллер является классом Java, опционально реализующим класс Initializable, который объявляется контроллером для файла FXML.
- Модель состоит из объектов домена, определенных на стороне Java, которые могут быть подключены к представлению через контроллер.

Установка компоновщика сцен

1. Загрузите самую последнюю версию Scene Builder с сайта Gluon, выбрав установщик для своей платформы или исполняемый банку.

2. С загруженным установщиком дважды щелкните, чтобы установить Scene Builder в вашей системе. Включена обновленная JRE.

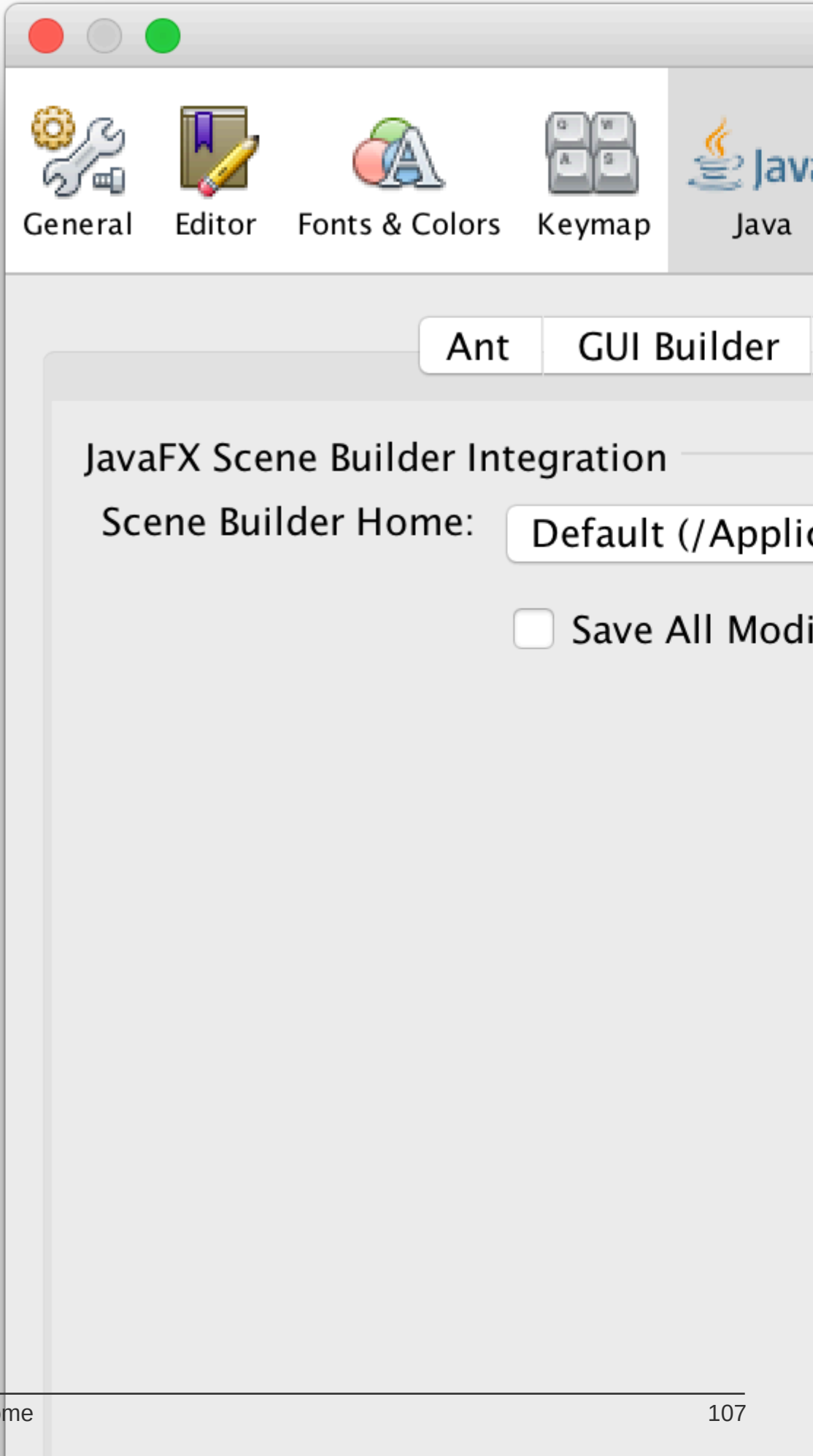
3. Дважды щелкните значок «Сценарий», чтобы запустить его как автономное приложение.

4. Интеграция с IDE

   Хотя Scene Builder является автономным приложением, он создает файлы FXML, которые интегрированы с проектом Java SE. При создании этого проекта в среде IDE

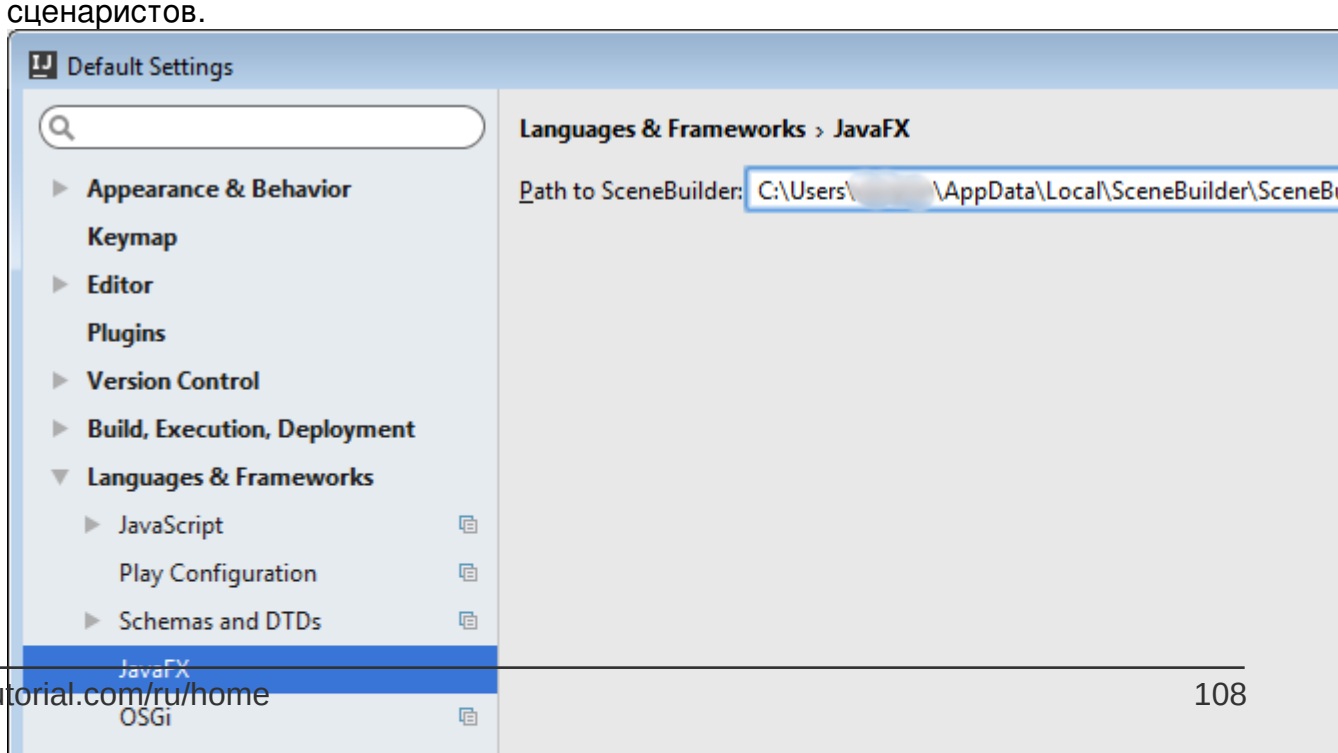
удобно включать ссылку на путь создания сценария, поэтому файлы FXML можно редактировать.

- NetBeans: в Windows перейдите в NetBeans-> Tools-> Options-> Java-> JavaFX. В Mac OS X перейдите в NetBeans-> Preferences-> Java-> JavaFX. Укажите путь для дома сценаристов.

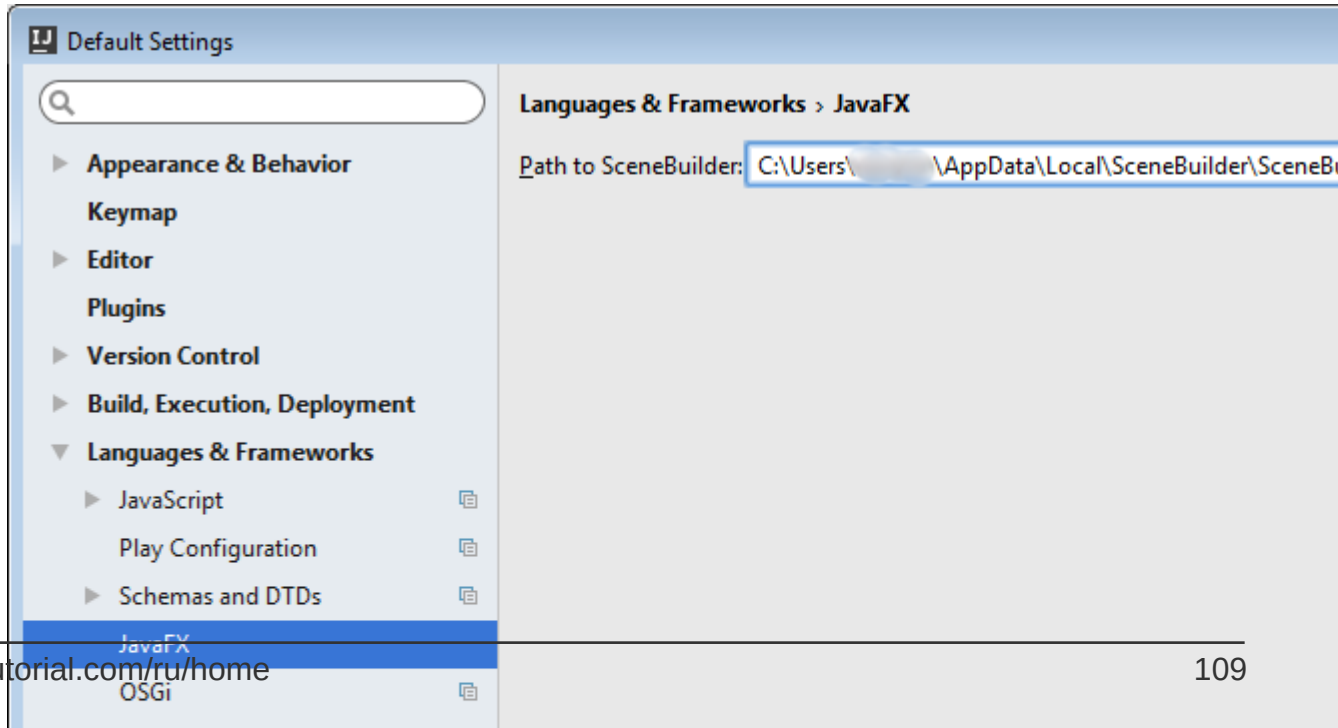

General
Editor
Fonts & Colors
Keymap
Java
Ant
GUI Builder
JavaFX Scene Builder Integration
Scene Builder Home:
Default (/Applic
Save All Modi

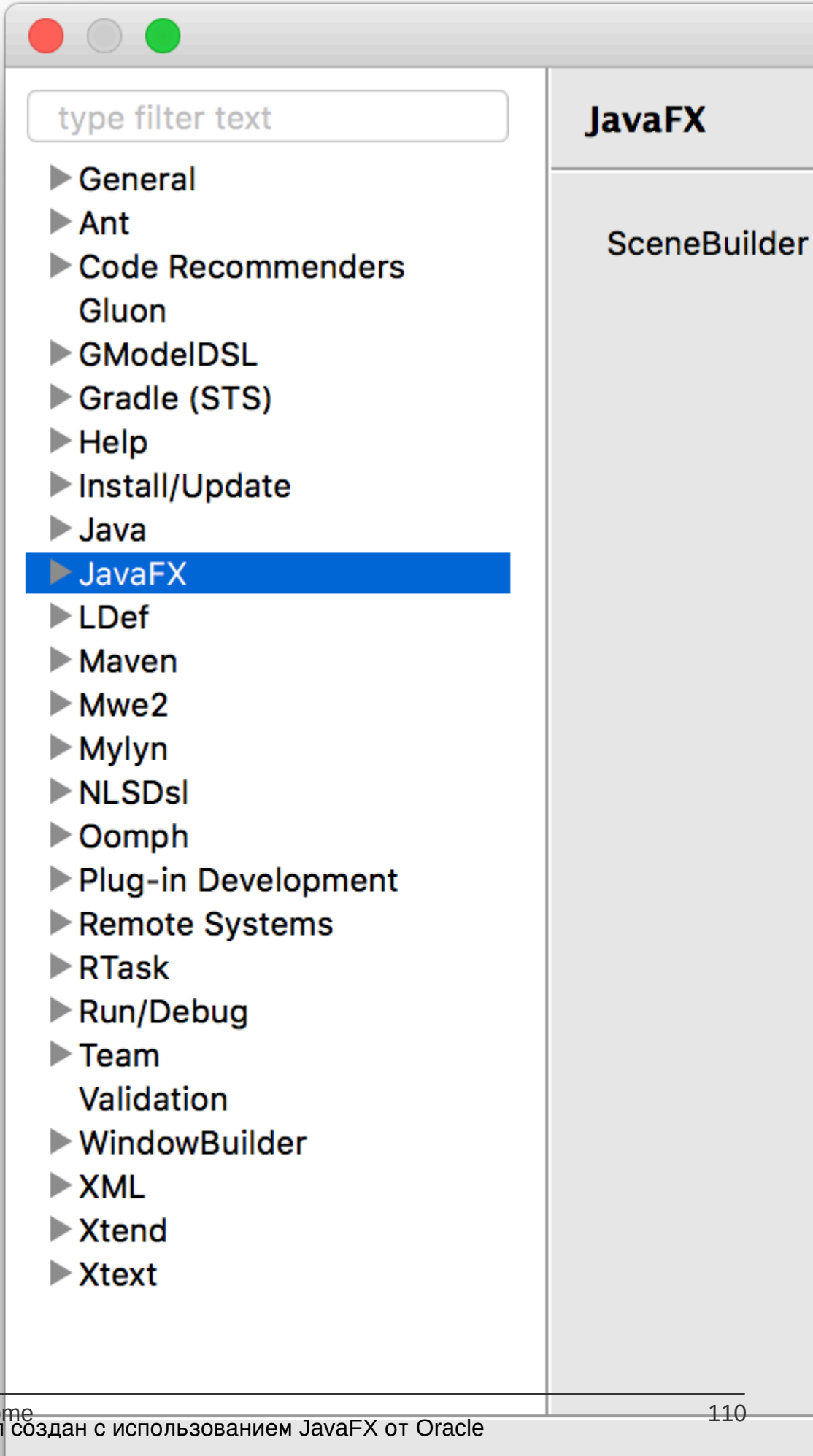
- IntelliJ: В Windows перейдите в IntelliJ-> Settings-> Languages & Frameworks-> JavaFX. В Mac OS X перейдите в IntelliJ-> Preferences-> Languages & Frameworks-> JavaFX. Укажите путь для дома сценаристов.

IntelliJ-> Settings-> Languages & Frameworks-> JavaFX. В Mac OS X перейдите в IntelliJ-> Preferences-> Languages & Frameworks-> JavaFX. Укажите путь для дома сценаристов.



- Eclipse: В Windows перейдите в Eclipse-> Window-> Preferences-> JavaFX. В Mac OS X перейдите в Eclipse-> Preferences-> JavaFX. Укажите путь для дома сценаристов.





Немного истории

Проект Scene Builder был создан с использованием JavaFX от Oracle

и является открытым исходным кодом в проекте OpenJFX.

Oracle предоставил двоичные файлы, вплоть до Scene Builder v 2.0, включая только функции JavaFX до выпуска Java SE 8u40, поэтому новые функции, такие как элементы управления `Spinner` , не включены.

Gluon взял на себя дистрибуцию бинарных выпусков, и обновленную версию Scene Builder 8+ можно загрузить для каждой платформы отсюда .

Он включает в себя последние изменения в JavaFX, а также последние улучшения и исправления ошибок.

Проект с открытым исходным кодом можно найти здесь, где могут быть созданы проблемы, запросы функций и запросы на тягу.

Оставшиеся двоичные файлы Oracle по-прежнему можно загрузить здесь .

Учебники

Учебники по сценариям можно найти здесь:

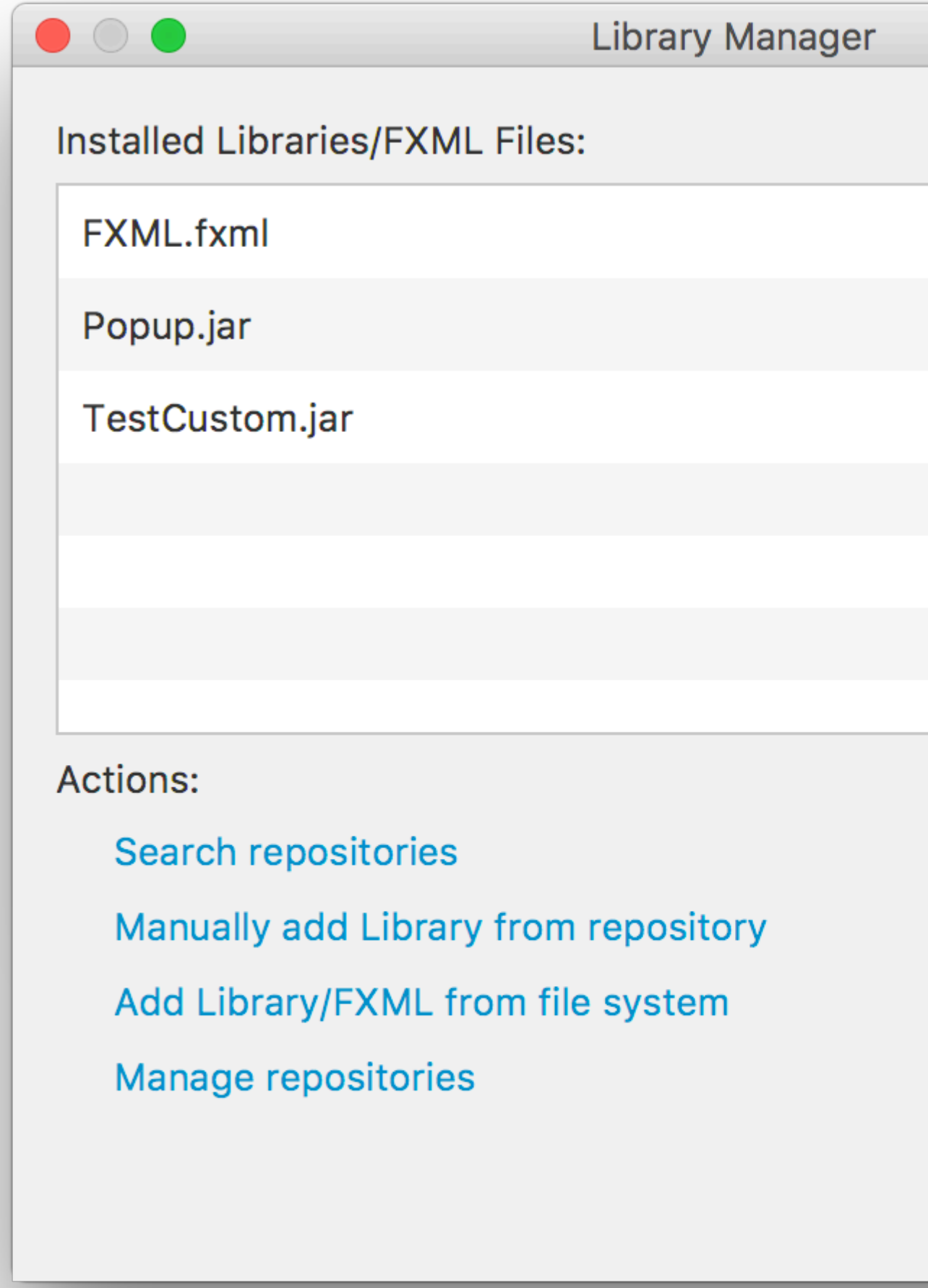
- Учебник Oracle Scene Builder 2.0

Учебники FXML можно найти здесь.

- Учебник Oracle FXML

Пользовательские элементы управления

Gluon полностью задокументировал новую функцию, которая позволяет импортировать сторонние банки с настраиваемыми элементами управления, используя Менеджер библиотек (доступный с момента создания Scene Builder 8.2.0).



Связанный тег: SceneBuilder

NetBeans (New Project -> JavaFX -> JavaFX FXML Application). Он содержит всего три файла:

Основной класс приложения

```
package org.stackoverflow;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class BasicApplication extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("BasicFXML.fxml"));

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Файл FXML

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

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="org.stackoverflow.BasicFXMLController">
    <children>
        <Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleButtonAction"
fx:id="button" />
        <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="label" />
    </children>
</AnchorPane>
```

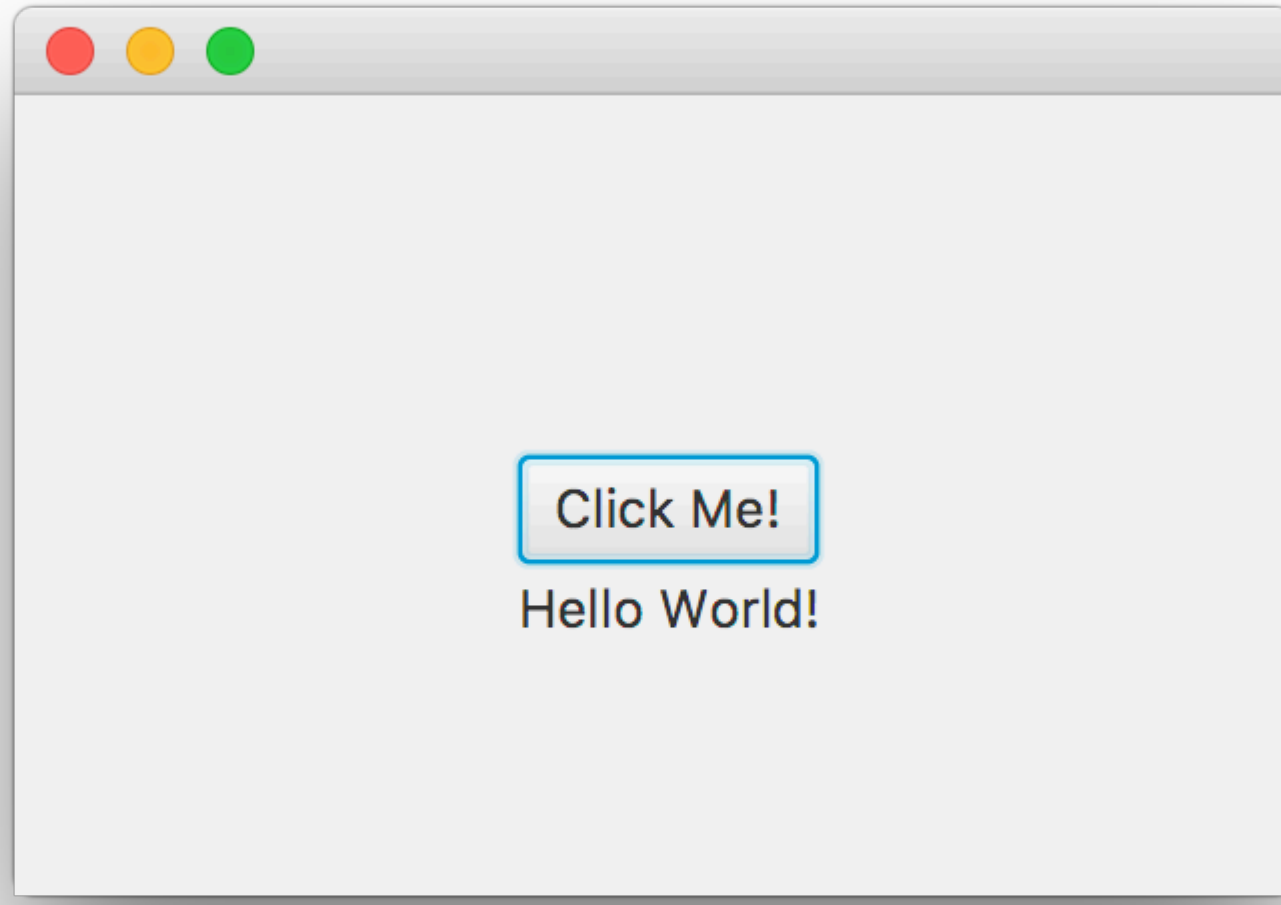
контроллер

```
package org.stackoverflow;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
```

```
public class BasicFXMLController {

    @FXML
    private Label label;

    public void initialize() {
        // TODO
    }

    @FXML
    private void handleButtonAction(ActionEvent event) {
        label.setText("Hello World!");
    }

}
```

Бежать

Создание и запуск проекта должно отображать небольшое окно с кнопкой:



Как это устроено

Вкратце, в основном классе Application, FXMLLoader будет загружать `basicFXML.fxml` из jar / classpath, как определено `FXMLLoader.load(getClass().getResource("BasicFXML.fxml"))` .

При загрузке `basicFXML.fxml` загрузчик найдет имя класса контроллера, как определено `fx:controller="org.stackoverflow.BasicFXMLController"` в FXML.

Затем загрузчик создаст экземпляр этого класса, в котором он попытается внедрить все объекты с `fx:id` в FXML и помечен аннотацией `@FXML` в классе контроллера.

В этом примере FXMLLoader создаст метку на основе `<Label ... fx:id="label"/>` , и она `@FXML private Label label;` экземпляр метки в `@FXML private Label label;` ,

Наконец, когда весь FXML загружен, FXMLLoader вызовет метод `initialize` контроллера, и

будет выполнен код, который регистрирует обработчик действий с помощью кнопки.

редактирование

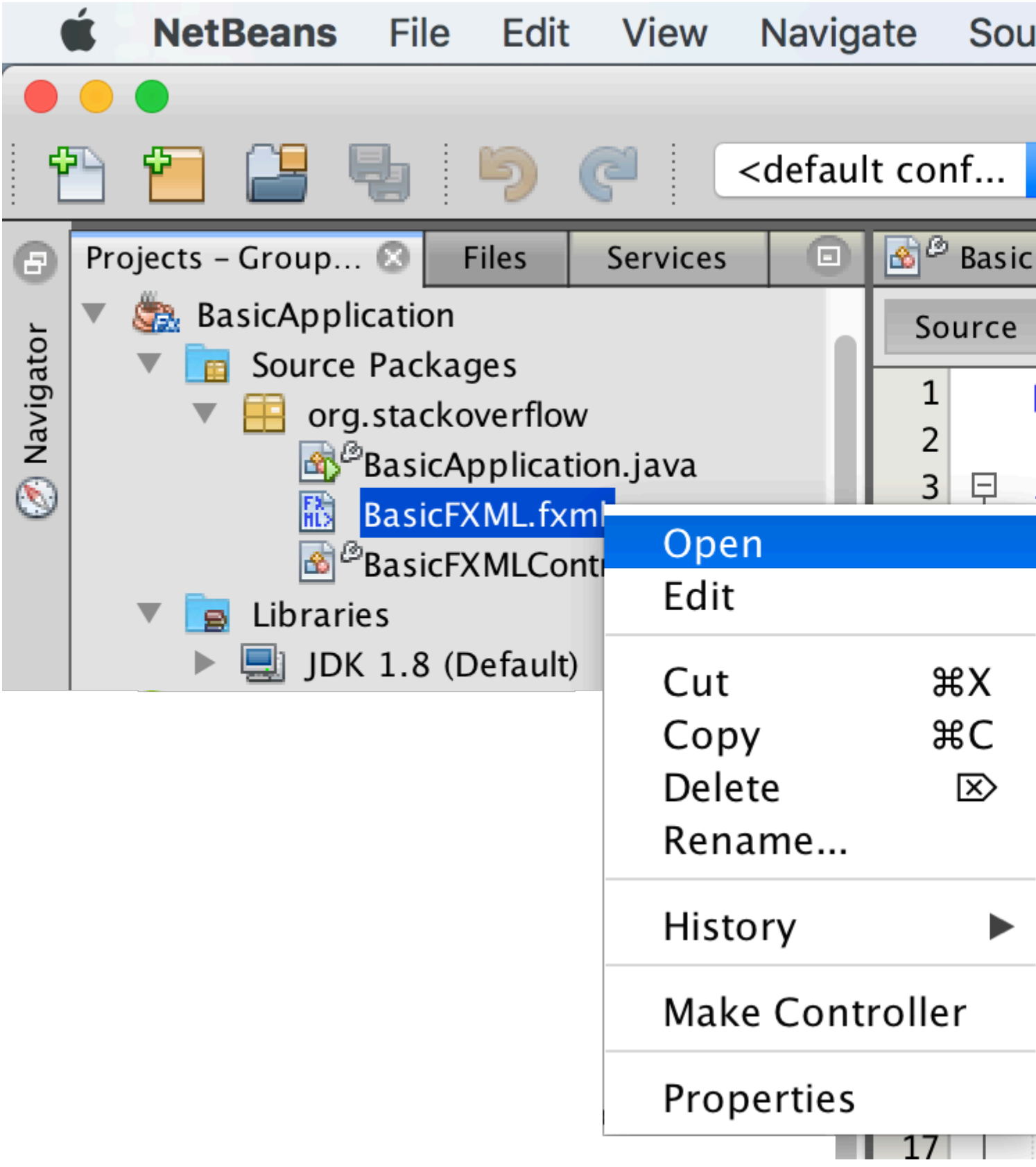
Хотя файл FXML можно редактировать в среде IDE, это не рекомендуется, так как среда IDE обеспечивает только базовую проверку синтаксиса и автозаполнение, но не визуальное руководство.

Лучшим подходом является открытие файла FXML с помощью Scene Builder, где все изменения будут сохранены в файле.

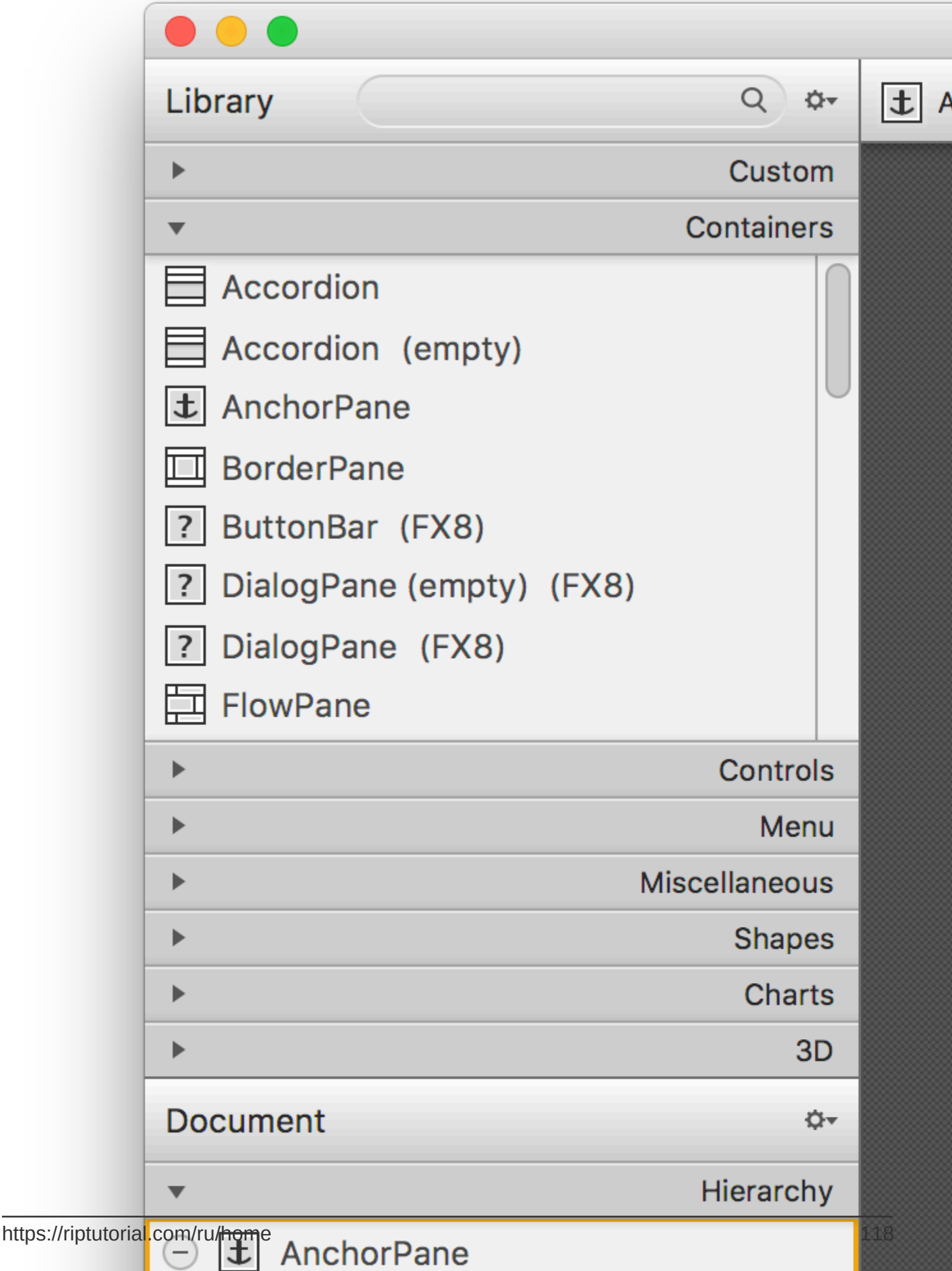
Scene Builder можно запустить, чтобы открыть файл:

Или файл можно открыть с помощью Scene Builder непосредственно из среды IDE:

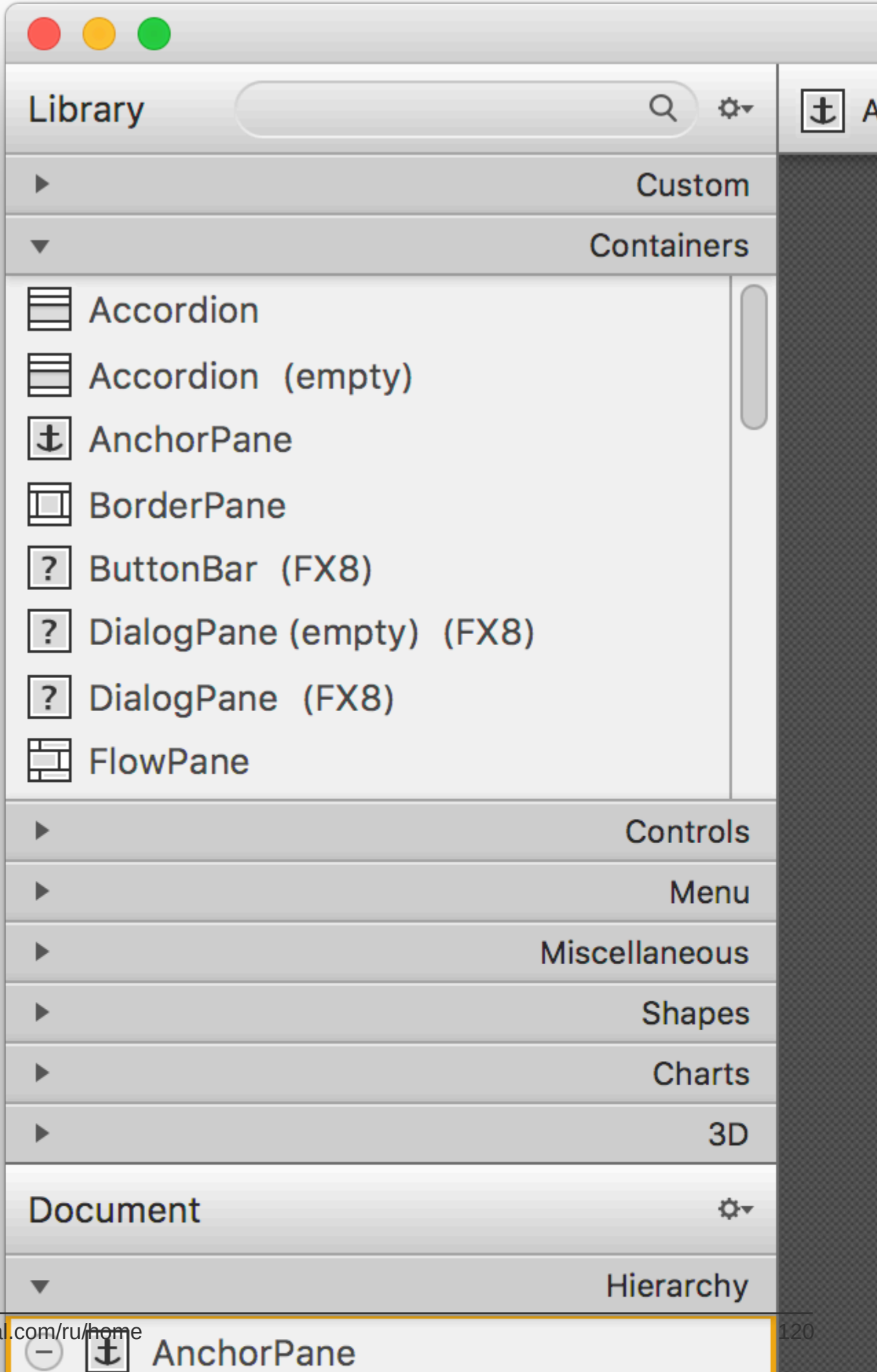
- Из NetBeans на вкладке проекта дважды щелкните файл или щелкните правой кнопкой мыши и выберите « `Open` .
- Из IntelliJ на вкладке проекта щелкните правой кнопкой мыши файл и выберите « `Open In Scene Builder` .
- Из Eclipse на вкладке проекта щелкните правой кнопкой мыши файл и выберите « `Open with Scene Builder` .



Если Scene Builder установлен правильно и его путь добавлен в среду IDE (см. Примечания ниже), он откроет файл:


Library
Custom
Containers
Accordion
Accordion (empty)
AnchorPane
BorderPane
ButtonBar (FX8)
DialogPane (empty) (FX8)
DialogPane (FX8)
FlowPane
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
AnchorPane

является `fx:id` . Его можно установить в панели « `Code` :


Library
Custom
Containers
Accordion
Accordion (empty)
AnchorPane
BorderPane
ButtonBar (FX8)
DialogPane (empty) (FX8)
DialogPane (FX8)
FlowPane
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
AnchorPane

Если при редактировании файла будут внесены изменения, отредактировав файл из среды IDE, они будут обновлены в Scene Builder.

Прочитайте Сценарий онлайн: https://riptutorial.com/ru/javafx/topic/5445/сценарий

глава 21: холст

Вступление

`Canvas` - это JavaFX- `Node` , представляемый как пустая прямоугольная область, которая может отображать изображения, фигуры и текст. Каждый `Canvas` содержит ровно один объект `GraphicsContext` , отвечающий за прием и буферизацию вызовов рисования, которые в конце отображаются на экране `Canvas` .

Examples

Основные формы

`GraphicsContext` предоставляет набор методов для рисования и заполнения геометрических фигур. Как правило, эти методы требуют, чтобы координаты передавались как их параметры, либо непосредственно, либо в виде массива `double` значений. Координаты всегда относятся к `Canvas` , происхождение которого находится в верхнем левом углу.

Примечание: `GraphicsContext` не будет рисоваться за пределами границ `Canvas` , т. Е. Попытка вывести за пределы области `Canvas` определяемую ее размером, и изменить ее размер впоследствии не даст результата.

Пример ниже показывает, как нарисовать три полупрозрачные заполненные геометрические фигуры, обозначенные черным штрихом.

```
Canvas canvas = new Canvas(185, 70);
GraphicsContext gc = canvas.getGraphicsContext2D();

// Set stroke color, width, and global transparency
gc.setStroke(Color.BLACK);
gc.setLineWidth(2d);
gc.setGlobalAlpha(0.5d);

// Draw a square
gc.setFill(Color.RED);
gc.fillRect(10, 10, 50, 50);
gc.strokeRect(10, 10, 50, 50);

// Draw a triangle
gc.setFill(Color.GREEN);
gc.fillPolygon(new double[]{70, 95, 120}, new double[]{60, 10, 60}, 3);
gc.strokePolygon(new double[]{70, 95, 120}, new double[]{60, 10, 60}, 3);

// Draw a circle
gc.setFill(Color.BLUE);
gc.fillOval(130, 10, 50, 50);
gc.strokeOval(130, 10, 50, 50);
```

Прочитайте холст онлайн: https://riptutorial.com/ru/javafx/topic/8935/холст

кредиты

S. No	Главы	Contributors
1	Начало работы с javafx	Community, CraftedCart, D3181, DVarga, fabian, Ganesh, Hendrik Ebbers, Petter Friberg
2	CSS	fabian
3	FXML и контроллеры	D3181, fabian, James_D
4	ScrollPane	Bo Halim
5	TableView	Bo Halim, fabian, GltknBtn
6	WebView и WebEngine	fabian, J Atkin, James_D, P.J.Meisch, Squidward
7	Windows	fabian, GltknBtn
8	Анимация	fabian, J Atkin
9	Диаграмма	Dth, James_D, Jinu P C
10	Диалоги	fabian, GltknBtn, Modus Tollens
11	Интернационализация в JavaFX	ItachiUchiha, Joffrey, Nico T, P.J.Meisch
12	кнопка	Dth, DVarga, J Atkin, Maverick283, Nico T, Squidward
13	Макеты	DVarga, fabian, Filip Smola, Jinu P C, Sohan Chowdhury, trashgod
14	Многопоточность	Brendan, fabian, GOXR3PLUS, James_D, Koko Essam, sazzy4o
15	пагинация	fabian, J Atkin
16	Переключатель	Nico T
17	печать	J Atkin, Squidward
18	Свойства и наблюдаемые	fabian
19	Связывание JavaFX	Alexiy

20	Сценарий	Ashlyn Campbell, José Pereda
21	холст	Dth