無料電子ブック

# 学習

# javafx

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#javafx

# 1: javafxをいめる

JavaFXは、いのデバイスでするリッチインターネットアプリケーションRIAだけでなく、デスクトップアプリケーションをしてするためのソフトウェアプラットフォームです。 JavaFXは、Java SEのGUIライブラリとしてSwingをきえることをしています。

ITは、リッチ・クライアント・アプリケーションの、、テスト、デバッグ、およびデプロイをうことができます。

JavaFXアプリケーションのは、スタイリングにCSSCascading Style Sheetsをしてカスタマイズすることができます JavaFXCSSを。FXMLファイルをしてをオブジェクトしてアプリケーションのややをにします FXMLおよびコントローラを 。シーンビルダはビジュアルエディタで、コードをすることなくUIのfxmlファイルをできます。

## バージョン

| バージョン | |
|---|---|
| JavaFX 2 | 20111010 |
| JavaFX 8 | 2014-03-18 |

## Examples

### インストールまたはセットアップ

> JavaFX APIは、Java SE Runtime EnvironmentJREとJava Development KitJDKのにされたとしてできます。 JDKはすべてのなデスクトッププラットフォームWindows、Mac OS X、およびLinuxでできるため、JDK 7にコンパイルされたJavaFXアプリケーションはすべてのなデスクトッププラットフォームでもします。 ARMプラットフォームのサポートもJavaFX 8でになりました。ARMJDKには、JavaFXのベース、グラフィックス、およびコントロールコンポーネントがまれています。

JavaFXをインストールするには、したバージョンのJava RuntimeとJava Developmentキットをインストールします。

JavaFXがするはのとおりです。

1. Java API。
2. FXMLとシーンビルダ。
3. WebView。
4. をスイングする。
5. みみのUIコントロールとCSS。

6. モデナテーマ。
7. 3Dグラフィックス。
8. キャンバスAPI。
9. API。
10. リッチテキストサポート。
11. マルチタッチサポート。
12. Hi-DPIサポート。
13. ハードウェアグラフィックスパイプライン。
14. メディアエンジン。
15. のアプリケーションデプロイメントモデル

## Hello Worldプログラム

のコードは、クリックに`String`をコンソールにするの`Button`をむなユーザーインターフェイスをします。

```
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);
    }

}
```

`Application`クラスはすべてのJavaFXアプリケーションのエントリポイントです。 `Application`は1つだけでき、これは

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

これにより、パラメータとしてされた`Application`クラスのインスタンスがされ、JavaFXプラットフォームがされます。

ここでプログラマーにとってなのはのとおりです。

1. に`launch`すると、 `Application`クラスこのは`HelloWorld` のしいインスタンスがされます。したがって、 `Application`クラスには、なしのコンストラクタがです。
2. された`Application`インスタンスにして`init()`がびされます。この、 `Application`からのデフォルトのはもしません。
3. `start`が`Appication`インスタンスにしてびされ、プライマリ`Stage` =ウィンドウがメソッドにされます。このメソッドは、JavaFXアプリケーションスレッドプラットフォームスレッドでにびされます。
4. アプリケーションは、プラットフォームがシャットダウンするとするまでされます。この、のウィンドウがじられたときにされます。
5. `stop`メソッドは、 `Application`インスタンスでびされます。この、 `Application`からのはもしません。このメソッドは、JavaFXアプリケーションスレッドプラットフォームスレッドでにびされます。

`start`では、シーングラフがされる。この、 `Button`と`StackPane`という2つの`Node`あり`StackPane`。

`Button`はUIの`Button`し、 `StackPane`は`Button`コンテナであり、そのをします。

これらの`Node`をする`Scene`がされます。に、 `Scene`がされるウィンドウである`Stage` `Scene`がされます。

オンラインでjavafxをいめるをむ https://riptutorial.com/ja/javafx/topic/887/javafxをいめる

# 2: CSS

- NodeClass / *ノードのクラスセレクタ* /
- .someclass / *クラスによるセレクタ* /
- #someId / * selector by id * /
- [selector1]> [selector2] / * selector2とするselector1とするノードののののセレクタ* /
- [selector1] [selector2] / selector2にするselector1とするノードののセレクタ* /

## Examples

### スタイリングにCSSをする

CSSはのにできます

- インライン `Node.setStyle`
- スタイルシートで
  - `Scene`
    - ユーザエージェントスタイルシートとしてここではしません
    - `Scene`「の」スタイルシートとして
  - `Node`

これにより、 `Nodes`スタイルなプロパティをできます。のはこれをしています

アプリケーションクラス

```
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;
}
```

## しいスタイルなプロパティをするの

JavaFX 8

のは、CSSからカスタム`Node`スタイルできるカスタムプロパティをするをしています。

ここで2つの`DoubleProperty`が`Rectangle`クラスにされ、CSSの`width`と`height`できます。

カスタムノードのスタイリングには、のCSSをできます。

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

## カスタムノード

```
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;
    }

}
```

オンラインでCSSをむ https://riptutorial.com/ja/javafx/topic/1581/css

# 3: FXMLとコントローラ

- xmlnsfx = " http://javafx.com/fxml " //

## Examples

### FXMLの

ボタンとラベルノードをむ`AnchorPane`をすなFXMLドキュメント

```
<?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>
```

このFXMLファイルのは、コントローラクラスにけられています。こののFXMLとコントローラクラスのけは、クラスをFXMLのルートの`fx:controller`のとしてすることによってわれます。fx `fx:controller="com.example.FXMLDocumentController"`コントローラクラスをすると、FXMLファイルでされているUIのユーザーアクションにじて、Javaコードをできます。

```
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`をしてFXMLファイルをロードすることができます

```
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();
    }

}
```

`load`メソッドはいくつかのアクションをし、するをするとです。このなでは、

1. `FXMLLoader`は、FXMLファイルをみんでします。ファイルにされているにするオブジェクトをし、されている`fx:id`をします。

2. FXMLファイルのルートは`fx:controller`をしているため、 `FXMLLoader`はするクラスのしいインスタンスをします。デフォルトでは、これはされたクラスでのないコンストラクタをびすことによってわれます。

3. フィールドがし、 `public` されないまたはき`@FXML` のいずれかであるフィールドをコントローラーにつ`fx:id`のは、するフィールドに「」されます。このではそう、そこにあるので、`Label`とFXMLファイル`fx:id="label"`としてされ、コントローラとフィールドが

   ```
   @FXML
   private Label label ;
   ```

   `label`フィールドは、 `FXMLLoader`によってされた`Label`インスタンスでされます。

4. イベントハンドラは、 `onXXX="#..."`プロパティがされたFXMLファイルのにされます。これらのイベントハンドラは、コントローラクラスのされたメソッドをびします。このでは、`Button`は`onAction="#handleButtonAction"`があり、コントローラはメソッドをしているため

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

ユーザーがボタンをすなどのアクションがボタンですると、このメソッドがびされます。このメソッドは`void`りでなければならず、イベントこのでは`ActionEvent`にするパラメータをするか、またはパラメータをすることができません。

5. に、コントローラクラスが`initialize`メソッドをしている、このメソッドがびされます。これは`@FXML`フィールドがされたにするので、このメソッドでにアクセスでき、FXMLファイルのにするインスタンスでされます。 `initialize()`メソッドは、パラメータをらないか、 `URL`と`ResourceBundle`とることができ`URL`。 のには、これらのパラメータは、によってされる`URL` FXMLファイルのをす、および`ResourceBundle`に`FXMLLoader`し`loader.setResources(...)`これらのいずれかがされていない、 `null`なるがあり`null`。

## ネストされたコントローラ

のコントローラをして、のFXMLにUIをするはありません。

`<fx:include>`タグをして、1つのfxmlファイルをのファイルにめることができます。インクルードされたfxmlのコントローラは、 `FXMLLoader`によってされたのオブジェクトとに、インクルードファイルのコントローラにできます。

これは、 `fx:id`を`<fx:include>`にすることによってわれます。このようにして、インクルードされたfxmlのコントローラは、`<fx:id value>Controller`フィールドにされ`<fx:id value>Controller`。

| fxid | のフィールド |
|---|---|
| foo | fooController |
| え42 | え42コントローラ |
| xYz | xYzController |

---

サンプル**fxmls**

カウンタ

これは`Text`ノードをつ`StackPane`をむfxmlです。このfxmlファイルのコントローラは、のカウンタのとカウンタのインクリメントをします。

**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;
    }

}
```

---

### fxmlをむ

```
<?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

インクルードされたfxmlのコントローラがこのコントローラにされます。ここでは、 `Button` `onAction`イベントのハンドラをしてカウンタをインクリメントします。

```
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();
    }

}
```

コードが`outer.fxml`とじパッケージのクラスからびされたとすると、fxmlsはのようにロードできます。

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

## ブロックをする

によっては、fxmlののオブジェクトのでをするがあることがあります。

これは*Define Blocks*がになるです

`<fx:define>`のは、にされたオブジェクトにはされません。

`<fx:define>`すべてのには、 `fx:id`がです。

このでされたオブジェクトは、 `<fx:reference>`をするか、またはバインディングをしてでできます。

`<fx:reference>`がつのをするためにすることができる`fx:id`された`<fx:reference>`とじをしてがされ`fx:id`でされるの`<fx:reference>`の`source`。

```
<?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>
```

## FXMLにデ―タをす - のコントロ―ラにアクセスする

のデ―タは、fxmlからロ―ドされたシ―ンにすがあります。

`fx:controller`をしてコントロ―ラをし、 `FXMLLoader`ロードにされる`FXMLLoader`インスタンスからロ―ドプロセスにされたコントロ―ラインスタンスをします。

デ―タをコントロ―ラインスタンスにし、それらのメソッドでデ―タをするメソッドをします。

**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>
```

コントロ―ラ

```
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);
    }

}
```

**fxml**のロードにされるコード

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

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

## FXMLへのデータのけし - コントローラインスタンスの

のデータは、fxmlからロードされたシーンにすがあります。

でfxmlをロードするためにする`FXMLLoader`インスタンスをしてコントローラをします。

fxmlをロードするに、コントローラにするデータがされていることをしてください。

この、fxmlファイルには`fx:controller`をめないでください。

**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>
```

コントローラ

```
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);
    }

}
```

**fxml**のロードにされるコード

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

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

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

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

## パラメータを**FXML**にす**- controllerFactory**をする

のデータは、fxmlからロードされたシーンにすがあります。

コントローラのをするコントローラファクトリをします。ファクトリでされたコントローラインスタンスにデータをします。

**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>
```

コントローラ

```
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);
    }

}
```

**fxml**のロードにされるコード

データ= "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();
```

---

これはにえるかもしれませんが、fxmlがとするコントローラークラスをできるはです。

## FXMLによるインスタンスの

のクラスをして、クラスのインスタンスをするをします。

JavaFX 8

`Person(@NamedArg("name") String name)`アノテーションは、 `@NamedArg`アノテーションができないためする`@NamedArg`があります。

```
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();
    }

}
```

fxmlをロードするに、 `Person`クラスがすでにされているものとします。

### にする

のfxmlのでは、importsセクションはされます。しかし、fxmlは

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

fxmlファイルでされるすべてのクラスをインポートするimportsセクションがきます。これらのインポートは、インポートにていますが、としてされます。 **`java.lang`**パッケージのクラスもインポートするがあり**`java.lang`**。

この、のインポートをするがあります。

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

JavaFX 8

## **`@NamedArg`**きコンストラクタ

すべてのパラメータをいてされているコンストラクタがある`@NamedArg`とのての`@NamedArg`はFXMLにする、コンストラクタは、これらのパラメータとにされるであろう。

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

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

がロードされた、のコンソールがられます。

```
Person(String)
```

## **args**コンストラクタなし

な`@NamedArg`きコンストラクタがない、パラメータをらないコンストラクタがされます。

`@NamedArg`アノテーションをコンストラクタからし、みみをみます。

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

これは、パラメータなしでコンストラクタをします。

```
Person()
setter
```

**fx:value**

fx:valueをして、そのをstatic valueOfメソッドにすことができますfx:valueこのメソッドは、Stringパラメ―タをし、するインスタンスをします。

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

```
valueOf
Person(String)
```

**fx:factory**

fx:factoryをすると、パラメ―タをらないのstaticメソッドをしてオブジェクトをできます。

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

```
createPerson
Person()
setter
```

**<fx:copy>**

fx:copyコンストラクタをびすことができます。のfx:idタグのsourceは、そのオブジェクトをパラメ―タとしてつコピ―コンストラクタをびします。

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

```
Person(Person)
getter
```

**fx:constant**

fx:constantはstatic finalフィ―ルドからをることをにします。

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

クラスをするときにされたJOHNをするだけなので、はされません。

プロパティの

fxmlのオブジェクトにデ―タをするはいくつかあります。

## <property> タグ

プロパティのをつタグは、インスタンスのにされるのとしてできます。このタグのは、セッター―をしてプロパティにりてられるか、プロパティのにされますみりのリスト/マップのプロパティ。

## デフォルトプロパティ

クラスに`@DefaultProperty`をけることができます。この、は、プロパティのをつをせずに、としてできます。

## property="value"

としてをし、としてをしてプロパティをりてることができます。これは、タグのとしてのをするのとじがあります。

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

## セッター―

プロパティは`static`セッター―をし`static`することもできます。これらは`setProperty`というの`static`メソッドで、を1のパラメ―タとして、を2のパラメ―タとしてします。これらのメソッドはどのクラスでもでき、のプロパティのわりに`ContainingClass.property`してできます。

、のが`String`ないり、するゲッター―メソッドつまり、メソッド `getProperty`というメソッドをセッター―とじクラスのパラメ―ター―としてをするメソッドをするがあります。

のメカニズムは、えばセッター―メソッドのパラメ―ター―にうように、りてにしいクラスのオブジェクトをするためにされます。

クラスがなは、そのものがされます。

それの、はのようにされます。

| ターゲットタイプ | されるソースs |
| --- | --- |
| `Boolean` 、 `boolean` | `Boolean.valueOf(s)` |
| `char` 、 `Character` | `s.toString.charAt(0)` |
| そののプリミティブまたはラッパー | ターゲットのなメソッド`valueOf(s.toString()) Number` sは、ラッパーの`valueOf(s.toString())` |
| `BigInteger` | `BigInteger.valueOf(s.longValue())`であるsで`Number` 、 `new BigInteger(s.toString())`それのは |
| `BigDecimal` | `BigDecimal.valueOf(s.doubleValue())`であるsで`Number` 、 `new BigDecimal(s.toString())`それのは |
| | `Double.valueOf(s.toString())`の`s.toString()`まれてい。、`Long.valueOf(s.toString())`それの |
| `Class` | クラスをせずにのスレッドのコンテキスト`ClassLoader`をしてびされた`Class.forName(s.toString())` |
| | `valueOf`メソッドのは、 sがそれぞれでまる`String`であれば、のにされた_られたすべてのの`String`にされます。 |
| その | される`static valueOf`タイプのマッチングパラメータしたtargetTypeのにおいて、 sまたはそのタイプのスーパークラスを |

このはにされておらず、されるがあります。

---

```
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;
    }

}
```

の`fxml`ファイルをみんだを`fxml`すると、

```
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>
```

オンラインでFXMLとコントローラをむ https://riptutorial.com/ja/javafx/topic/1580/fxmlとコントローラ

# 4: JavaFXでの

## Examples

### リソースバンドルのロード

JavaFXは、ユーザーインターフェイスをするなをします。 FXMLファイルからビューをするに、`FXMLLoader`にリソースバンドルをすることができます。

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

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

このされたバンドルは、 `%`でまるFXMLファイルのすべてのテキストをにするためにされます。プロパティファイル`strings_en_UK.properties`にのがまれているとします。

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

FXMLにのようなボタンがある

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

`ui.button.text`キーのがにされます。

### コントローラ

リソースバンドルには、ロケールのオブジェクトがまれています。バンドルは`FXMLLoader`にすことができます。コントローラは`Initializable`インターフェイスをし、 `initialize(URL location, ResourceBundle resources)`メソッドをオーバーライドするがあります。このメソッドの2のパラメータは`ResourceBundle`これはFXMLLoaderからコントローラにされ、コントローラがテキストをさらにしたり、のロケールのをするためにできます。

```
public class MyController implements Initializable {

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

### アプリケーションのにをにりえる

このでは、アプリケーションのにをにりえることができるJavaFXアプリケーションをするをしま

す。

このでされているメッセージバンドルファイルはのとおりです。

### 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}
```

なアイデアは、ユーティリティクラスI18Nをつことですとして、これはシングルトンにされるかもしれません。

```
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;
    }
}
```

このクラスには、JavaFX `ObjectProperty`ラップされたJava `Locale`オブジェクトであるフィールド`locale`があり、このプロパティにしてバインディングをできます。のメソッドは、JavaFXプロパティをおよびするためのなメソッドです。

`get(final String key, final Object... args)`は、 `ResourceBundle`からののメッセージにされるコアメソッドです。

`createStringBinding`という2つのメソッドは、 `locale`フィールドにバインドされた`StringBinding`をし、 `locale`プロパティがされるたびにバインディングがされるようにし`locale`。のは、の`get`メソッドをしてメッセージをおよびするをし、2のは`Callable`にされ、しいをするがあります。

の2つのメソッドは、JavaFXコンポーネントをするメソッドです。のメソッドは、 `Label`をするためにされ、バインディングのために`Callable`をします。 2は`Button`をし、キーをしてStringバインディングをします。

もちろん、 `MenuItem`や`ToolTip`ようにくのなるオブジェクトをすることができ`ToolTip`が、これらの2つのでです。

このコードは、このクラスがアプリケーションでどのようにされるかをしています。

```
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);
    }
}
```

このアプリケーションは、 `I18N`クラスによってされた`StringBinding`をする3つのなるをしています。

1. ウィンドウタイトルは`StringBinding`してバインドされます。
2. ボタンはヘルパーメソッドをメッセージキーとにします
3. ラベルは`Callable`ヘルパーメソッドをします。この`Callable`は`I18N.get()`メソッドをして、スイッチののをむきのされたをします。

ボタンをクリックすると、カウンタがし、 `I18N`のロケールプロパティがされ、バインディングのがトリガされ、UIのがしいにされます。

オンラインでJavaFXでのをむ https://riptutorial.com/ja/javafx/topic/5434/javafxでの

# 5: JavaFXバインディング

## Examples

なプロパティバインディング

JavaFXには、あるプロパティをのプロパティにバインドするをするバインディングAPIがあります。つまり、あるプロパティのがされるたびに、バインドされたプロパティのがにされます。バインディングの

```
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
}
```

また、、、などをしてプリミティブプロパティをバインドすることもできます。

```
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'
}
```

のオブジェクトをSimpleObjectPropertyにれることができます

```
SimpleObjectProperty<Color> color=new SimpleObjectProperty<>(Color.web("45f3d1"));
```

バインディングをすることはです。この、プロパティはにします。

```
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'
}
```

オンラインでJavaFXバインディングをむ https://riptutorial.com/ja/javafx/topic/7014/javafxバインディング

# 6: ScrollPane

き

ScrollPaneは、コンテンツのをするコントロールです。このビューはさまざまなでされます。 インクリメント/デクリメントボタン/マウスホイールをしてコンテンツをにします。

## Examples

### Aコンテンツのサイズ

コンテンツのサイズは、ScrollPaneコンテナのサイズとじになります。

```
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コンテンツのサイズ

コンテンツのサイズは、ビューをすることによってえるのおよびのコンテンツをえるされたにじてされます。

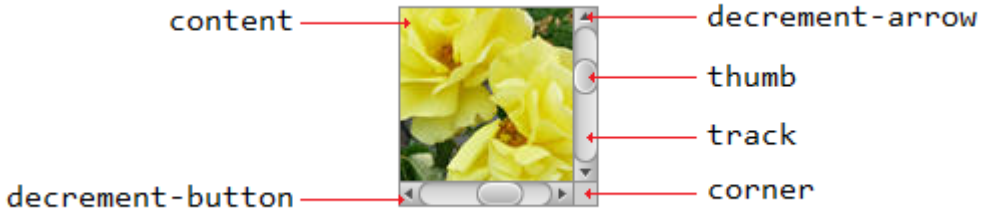
```
import javafx.scene.control.ScrollPane;   //Import the ScrollPane
import javafx.scene.control.ScrollPane.ScrollBarPolicy; //Import the ScrollBarPolicy
import javafx.scene.layout.Pane;

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
```

ここでは、のメソッドsetFitToWidth / setFitToHeightはありません。

## ScrollPaneのスタイル

ScrollPaneのは、「 *CSS* 」というをち、いくつかのコントロールの「 プロパティ 」をし、もちろん「 」をつことで、にできます。

### AScrollPaneをする




### BCSSプロパティ

```
.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{}
```

オンラインでScrollPaneをむ https://riptutorial.com/ja/javafx/topic/8259/scrollpane

# 7: TableView

## Examples

### 2のサンプルTableView

テーブルアイテム

のクラスは、name `String` とsize `double` の2つのプロパティをみます。どちらのプロパティもJavaFXプロパティにラップされているため、 `TableView`はをできます。

```
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;
    }

}
```

サンプルアプリケーション

このアプリケーションは2の`TableView`をしています。 1つはのためのもので、もう1つは`Person`サイズのものです。 `Person`の1つをすると、 `TableView`ののの`TextField`データがされ、ユーザーはデータをできます。がコミットされると、 `TableView`がにされます。

`TableView`されたすべての`TableColumn` `TableView` 、 `cellValueFactory`がりてられます。このファクトリは、の `Person` をセルにするがあるをむ`ObservableValue`にし、 `TableView`がこののをリッスンできるようにします。

```
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`は、 `TableColumn cellValueFactory`としてできます。のパターンとするメソッドにアクセスするためにリフレクションをして、 `TableView`アイテムからデータをします。

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

データをするためにされるメソッドのは、 `PropertyValueFactory`コンストラクターパラメーターでまります。

- プロパティメソッドこののメソッドは、データをむ`ObservableValue`をすことがされます。がられます。それらは、 `<constructor parameter>Property`パターンのパターンとし、 `<constructor parameter>Property`をるがありません。
- **Getter**メソッドこののメソッドは、をしますのでは`String`。 メソッドは、パターン `get<Constructor parameter>`とするがあり`get<Constructor parameter>`。 ここで`<Constructor parameter>`はでまります 。このメソッドはパラメータをるべきではありません。

メソッドのサンプル

| コンストラクタパラメータなし | プロパティメソッドの | ゲッターメソッドの |
|---|---|---|
| foo | fooProperty | getFoo |
| fooBar | fooBarProperty | getFooBar |

| コンストラクタパラメータなし | プロパティメソッドの | ゲッターメソッドの |
|---|---|---|
| XYZ | XYZプロパティ | getXYZ |
| listIndex | listIndexProperty | getListIndex |
| | aValueProperty | getAValue |

## アイテムにじて**TableCell**のをカスタマイズする

には、セルの`toString`とはなるがされることがあります。この、 `TableColumn`の`cellFactory`によってされた`TableCell`は、についてレイアウトをするようにカスタマイズされています。

な `TableView`は、UIにされる`TableCell`のみをし`TableCell`。 セルのアイテムはし、になることさえあります。プログラマーは、されたときにアイテムがされたときにわれた`TableCell`をにするようにするがあり`TableCell`。 それの、コンテンツは「それがしていない」セルにされることがあります。

のでは、をすると、 `ImageView`されるとにテキストがされます。

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

が`null`たり、セルがになったりすると、を`null`すことでがりされ`null`。

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

---

のは、 `TableCell`テキストにえて、をしてい`TableCell`。

`updateItem`メソッドは、 `Cell`のがされるたびにびされます。このメソッドをオーバーライドすると、にしてセルのをできます。リスナーをセルの`itemProperty()`にするもあり`TableCell`が、くの、`TableCell`がされています。

アイテムタイプ

```
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;
    }

}
```

アプリケーションクラスのコード

```
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);
    }

}
```

セル クラス

```
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());
        }
    }
}
```

## Tableviewにボタンをする

`setCellFactory(Callback value)`メソッドのをして、Tableviewにボタンまたはのjavafxコンポーネントをできます。

サンプルアプリケーション

このアプリケーションでは、TableViewにボタンをします。このボタンをクリックすると、ボタンとじのデータがされ、そのがされます。

`addButtonToTable()`メソッドでは、 `cellFactory`コールバックがするにボタンをします。びしなcellFactoryをし、オーバーライド`call(...)`メソッドをして、ボタンで`TableCell`をし、にこの

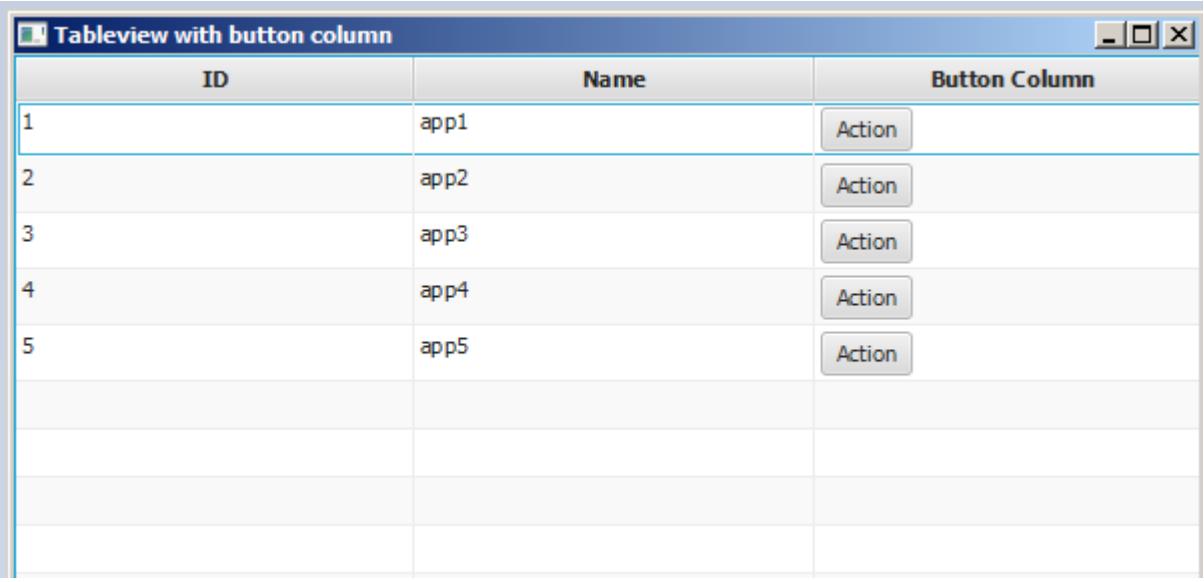
`cellFactory`を`setCellFactory(..)`メソッドにします。サンプルでは、これは`colBtn.setCellFactory(cellFactory)`です。 SSCCEはのりです

```
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;
        }

    }
}
```

スクリーンショット




オンラインでTableViewをむ https://riptutorial.com/ja/javafx/topic/2229/tableview

# 8: WebViewとWebEngine

WebViewは、JavaFXコンポーネントツリーにされたJavaFXノードです。 WebEngineをし、WebEngineのコンテンツをします。

WebEngineはにのをうなブラウザエンジンです。

## Examples

ページのみ

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

WebViewはWebEngineりのUIシエルWebEngine。 UIとののためのほぼすべてのコントロールは、WebEngineクラスをしてわれます。

### WebViewのページをする

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

はにエントリのリストです。エントリはされたページをし、URL、タイトル、およびページがにされたなどのページへのアクセスをする。

リストは、 getEntries()メソッドをしてできます。 WebEngineがWebをナビゲートすると、とエントリのするリストがわります。リストはブラウザのにじてまたはすることがあります。これらのは、リストがするObservableList APIによってくことができます。

しているページにけられているエントリのインデックスは、 currentIndexProperty()によってされます。のインデックスは、 go(int)メソッドをしてののエントリにナビゲートするためにできます。 maxSizeProperty()は、ヒストリリストのサイズであるヒストリサイズをします

は、 Webのリストをしてするのです。

ComboBox comboBoxは、アイテムをするためにされます。してListChangeListenerWebHistory ComboBoxにされますWebHistory。 ComboBoxは、したページにリダイレクトするEventHandlerあります。

```
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();
    }
});
```

されたWebページからJavascriptアラートをJavaアプリケーションログにします。

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

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

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

ウェブページのJavaアプリケーションとJavascriptの

WebViewをしてのカスタムWebページをし、このWebページにJavascriptがまれている、JavaプログラムとWebページのJavascriptとのでのをするがあります。

このでは、このようなをするをします。

Webページには、フィールドとボタンがされます。ボタンをクリックすると、フィールドのがJavaアプリケーションにられ、されます。、はJavascriptにられ、はWebページにされます。

なは、JavascriptからJavaへののために、WebページにされたJavaでオブジェクトがされることです。もうのでは、オブジェクトがJavaScriptでされ、Webページからされます。

のコードはJavaのをしています。

```
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());
            }
        }
    }
}
```

ページがロードされると、 JavaConnectorオブジェクトクラスによってされ、フィールドとしてされますがのびしによってWebページにされます。

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

javascriptConnectorオブジェクトは、のWebページからされます。

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

toLowerCase(String)からメソッドJavaConnectorびされ、にされたされたをしてりされる javascriptConnectorオブジェクト。

そして、これはhtmlとjavascriptのコードです

```
<!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>
```

`sendToJava`は、Javaコードによってされた`JavaConnector`のメソッドをびします。

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

`javascriptConnector`をするためにJavaコードによってびされるは、 `jsConnector`オブジェクトをします。

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

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

JavaとJavascriptののびしのは、にされません。なとのは、 JSObject APIのドキュメントをしてください 。

オンラインでWebViewとWebEngineをむ https://riptutorial.com/ja/javafx/topic/5156/webviewとwebengine

# 9: Windows

## Examples

### しいウィンドウをする

しいウィンドウにコンテンツをするには、 `Stage`をするがあります。および、 `show`オブジェクトまたは`showAndWait`オブジェクトを`Stage`オブジェクトで`showAndWait`があります。

```
// 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();
```

このコーードは、JavaFXアプリケーションスレッドでするがあります。

### カスタムダイアログの

くのコンポーネントをむカスタムダイアログをし、くのをできます。オーナーステージでは2のステージのようにるいます。
のでは、メインステージtableviewにをし、されたダイアログAddingPersonDialogにをするアプリケーションをします。 SceneBuilderによってされたGUIですが、なJavaコードでできます。

サンプルアプリケーション

**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

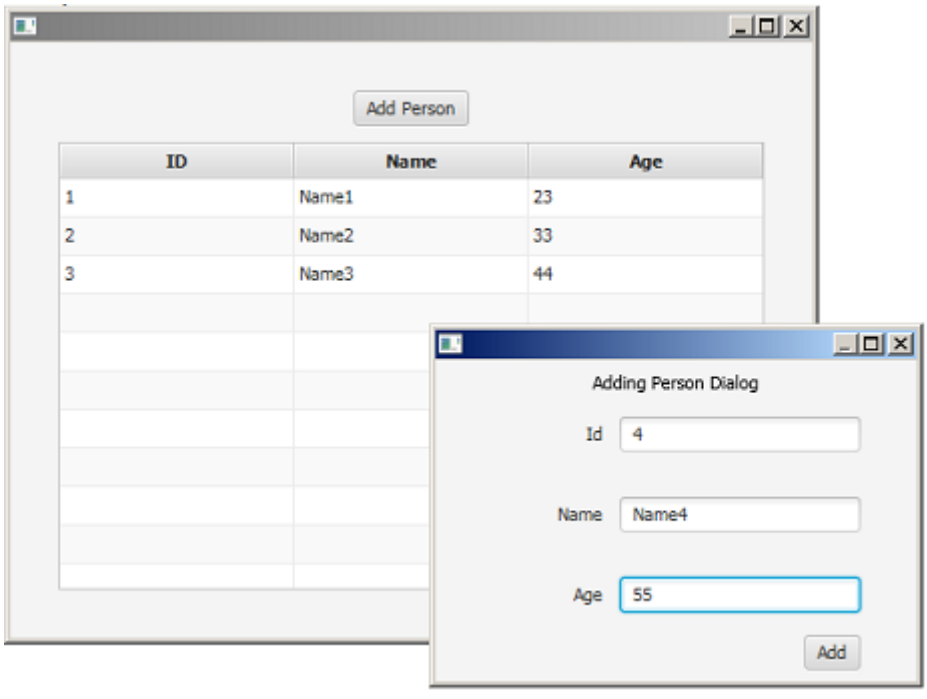
```
<?xml version="1.0" encoding="UTF-8"?>

<?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();
    }

}
```




スクリーンショット

## カスタムダイアログの

くのコンポーネントをむカスタムダイアログをし、くのをできます。オーナーステージでは2のステージのようにるいます。
のでは、メインステージtableviewにをし、されたダイアログAddingPersonDialogにをするアプリケーションをします。 SceneBuilderによってされたGUIですが、なJavaコードでできます。

サンプルアプリケーション

### 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?>
```

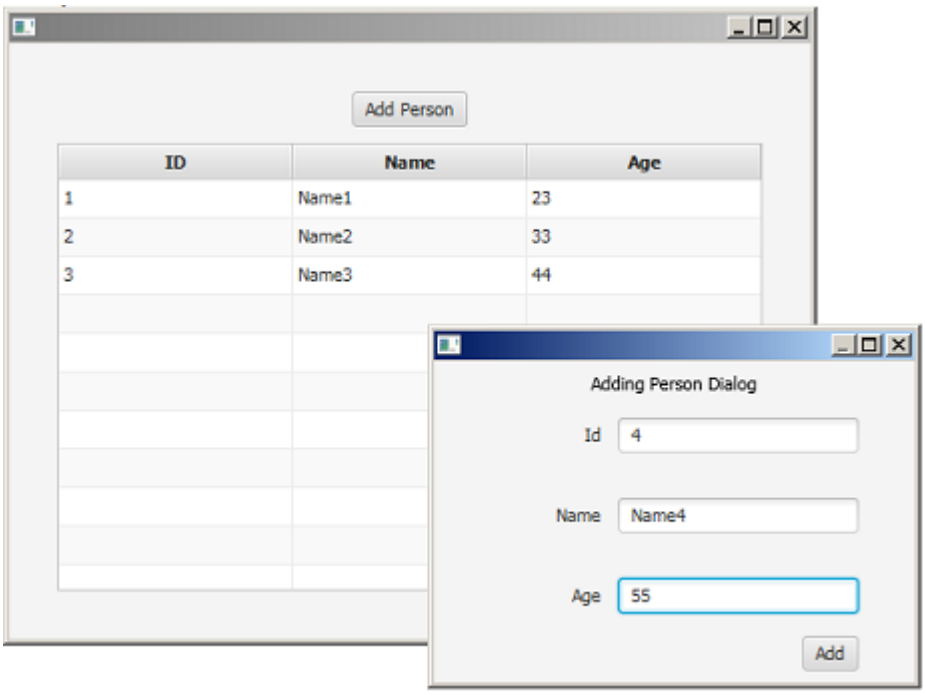
```
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>

<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();
    }

}
```

スクリーンショット




オンラインでWindowsをむ https://riptutorial.com/ja/javafx/topic/1496/windows

# 10: アニメーション

## Examples

タイムラインをしたプロパティのアニメーション

```
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();
```

JavaFXでアニメーションをするもでなは、 `Timeline`クラスです。タイムラインは、 `KeyFrame`アニメーションののとしてしてし`KeyFrame`。 この、 `0 seconds` に`translateXProperty`をゼロにするがあり、 `2 seconds` にプロパティを`80`するがあることが`80`ます。また、アニメーションをにしたり、するかといったのをうこともできます。

タイムラインはのプロパティをにアニメートできます

```
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
```

このアニメーションは`Y`プロパティを`0` プロパティのから`10`を1にとり、3で`90`でします。タイムラインののではないにもかかわらず、 `Y`がアニメーションをすると、 `Y`はゼロにります。

オンラインでアニメーションをむ https://riptutorial.com/ja/javafx/topic/5166/アニメーション

# 11: キャンバス

き

`Canvas`は、、、およびテキストをできるののとしてされるJavaFX `Node`です。`Canvas`は、びしのとバッファリングをする`GraphicsContext`オブジェクトが1つまれています。びしは、に`Canvas`によってにされます。

## Examples

`GraphicsContext`は、をしてりつぶすためのメソッドのセットをします。、これらのメソッドでは、または`double`ののので、としてパラメータをすがあります。は、 `Canvas`にしています。 `Canvas`はです。

`GraphicsContext`は`Canvas`のにされません。つまり、サイズでされた`Canvas`にし、でサイズをするとがされません。

のは、いストロークでがかれた3つののりつぶしたをくをしています。

```
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);
```

オンラインでキャンバスをむ https://riptutorial.com/ja/javafx/topic/8935/キャンバス

# 12: シーンビルダー

## き

JavaFX Scene Builderはビジュアルレイアウトツールで、コーディングなしでJavaFXアプリケーションのユーザーインターフェイスをすばやくできます。 FXMLファイルをするためにされます。

JavaFX Scene Builderはビジュアルレイアウトツールで、コーディングなしでJavaFXアプリケーションのユーザーインターフェイスをすばやくできます。ユーザーは、UIコンポーネントをワークエリアにドラッグアンドドロップし、プロパティをしたり、スタイルシートをしたり、のレイアウトのFXMLコードをバックグラウンドでにしたりできます。その、アプリケーションのロジックにUIをバインドすることによって、Javaプロジェクトとできるfxmlファイルがされます。

モデルビューコントローラMVCのから

- ユーザーインターフェイスのをむFXMLファイルがビューです。
- コントローラはJavaクラスであり、FXMLファイルのコントローラとしてされているInitializableクラスをすることもできます。
- このモデルは、Javaでされたドメインオブジェクトでされ、コントローラをしてビューにできます。

## シーンビルダーのインストール

1. Scene BuilderのバージョンをGluonのWebサイトからダウンロードして、しているプラットフォームのインストーラまたはなjarファイルをします。

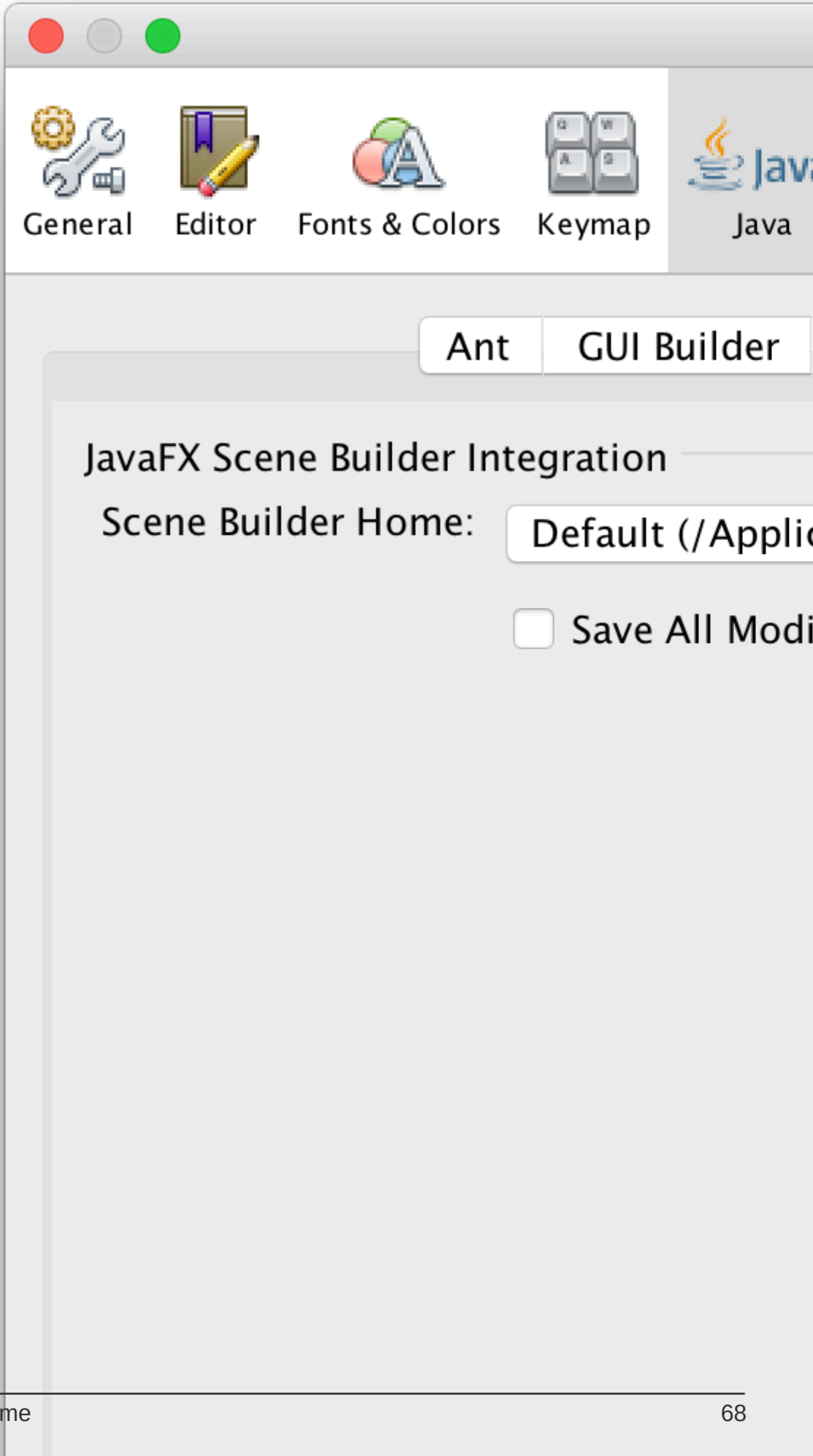
2. インストーラをダウンロードしたで、システムにScene Builderをダブルクリックしてインストールします。されたJREがまれています。

3. シーンビルダアイコンをダブルクリックしてスタンドアロンアプリケーションとしてします。

4. IDE

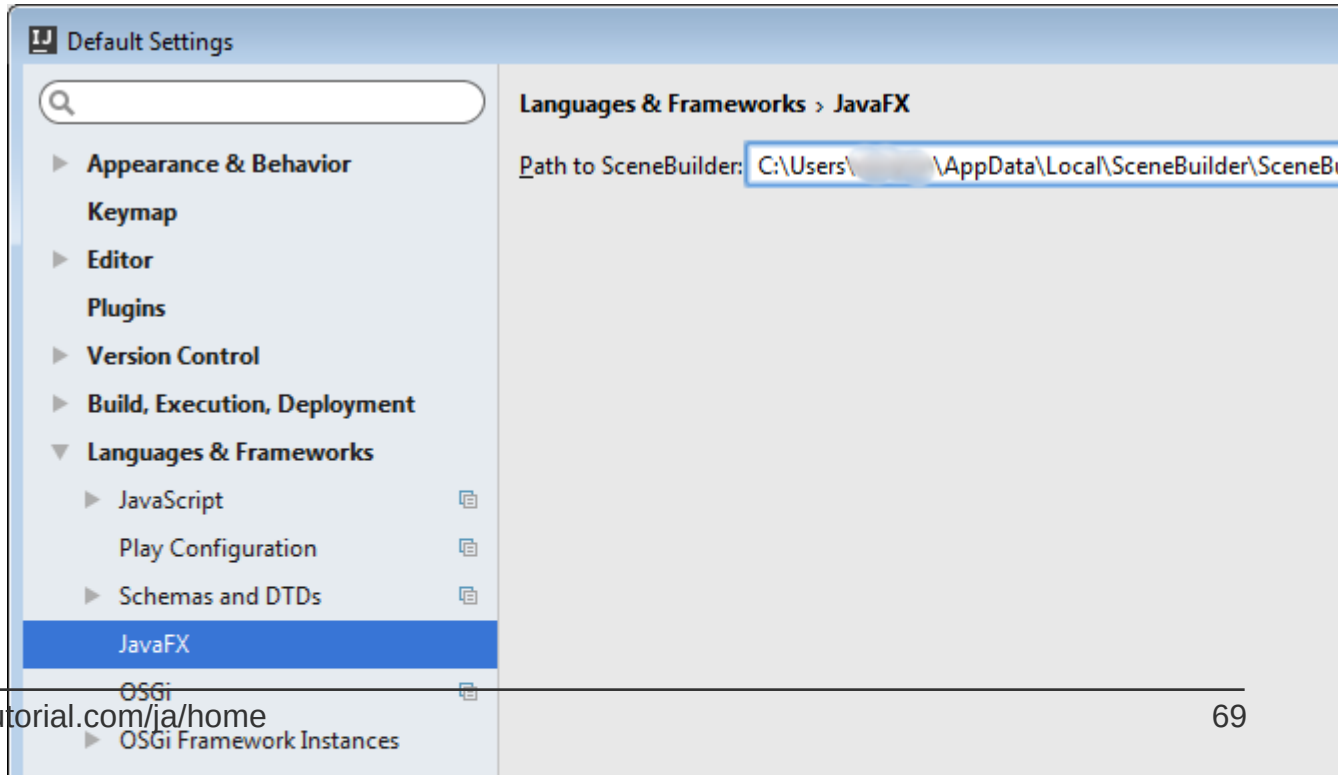
   シーンビルダはスタンドアロンアプリケーションですが、Java SEプロジェクトとされたFXMLファイルをします。このプロジェクトをIDEでするときは、シーンビルダーのパスへのリンクをめるとです.FXMLファイルをすることができます。

   - NetBeansWindowsでは、NetBeans-> Tools-> Options-> Java-> JavaFXにします。Mac OS Xでは、NetBeans-> Preferences-> Java-> JavaFXにします。シーンビルダホームのパスをします。

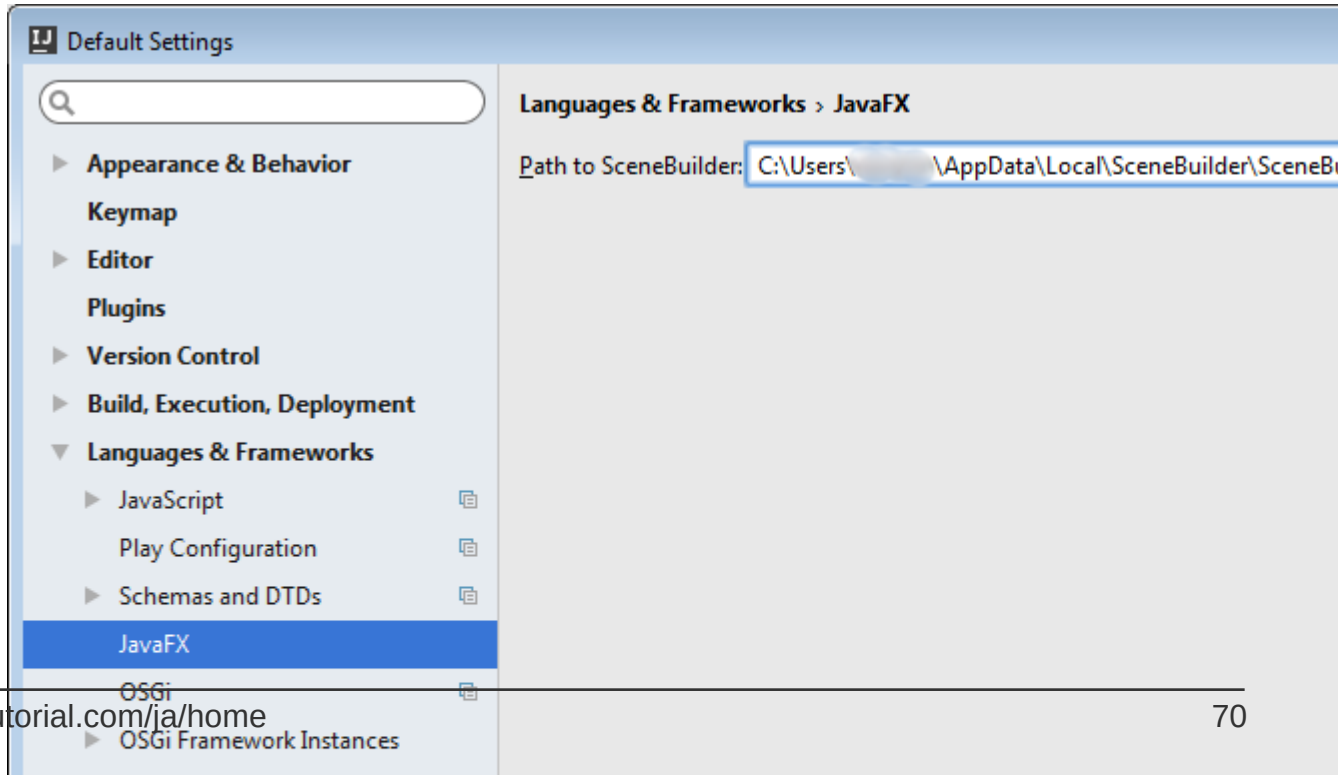

General
Editor
Fonts & Colors
Keymap
Java
Ant
GUI Builder
JavaFX Scene Builder Integration
Scene Builder Home:
Default (/Applic
Save All Modi

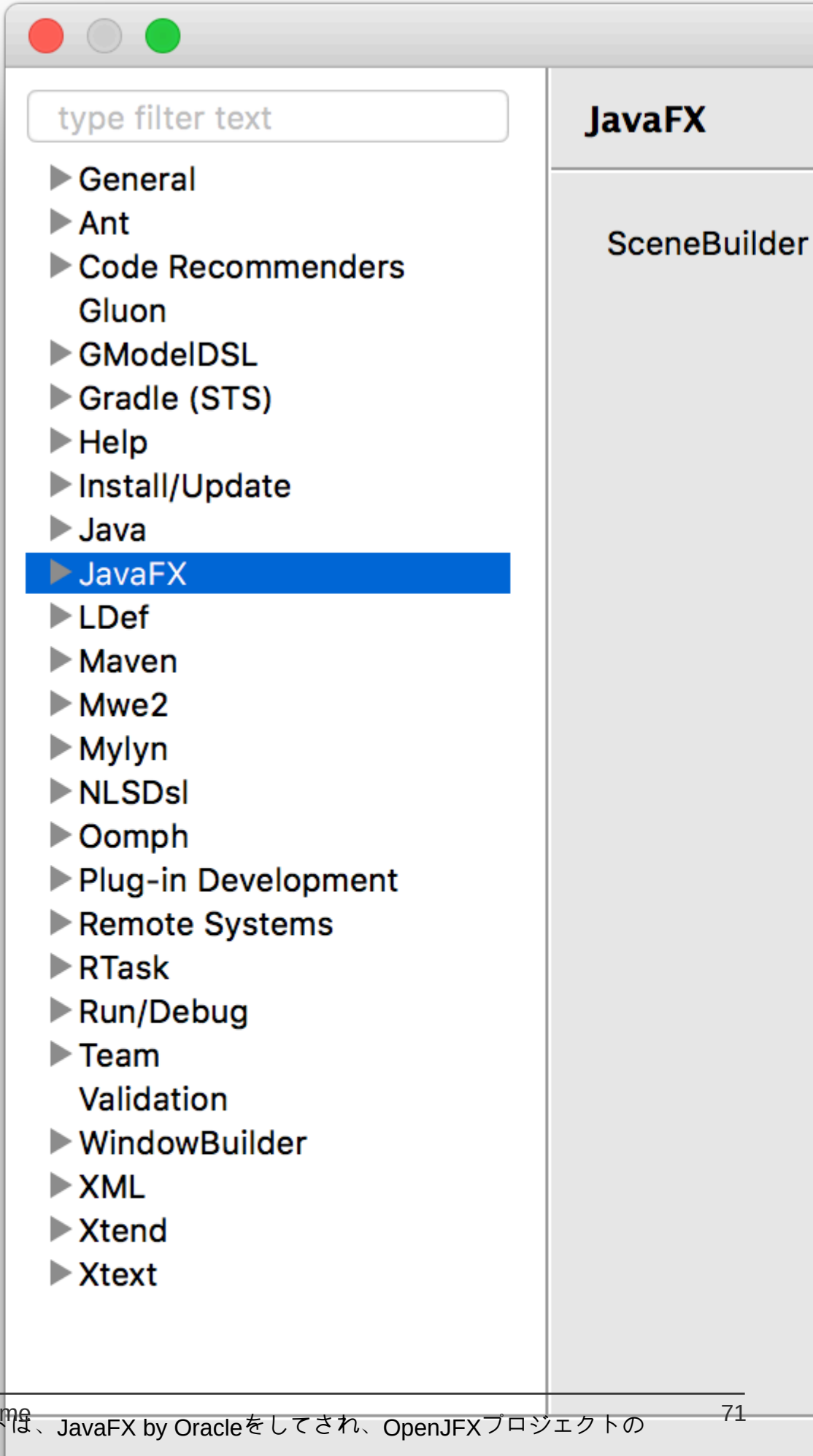
- IntelliJWindowsではIntelliJ-> Settings-> LanguagesFrameworks-> JavaFXにします。Mac OS Xでは、[IntelliJ] - > [] - > [とフレームワーク] - > [JavaFX]をします。シーンビルダホームのパスをします。

Mac OS Xでは、[IntelliJ] - > [] - > [とフレームワーク] - > [JavaFX]をします。シーンビルダホームのパスをします。



- EclipseWindowsでは、Eclipse-> Window-> Preferences-> JavaFXにします。 Mac OS Xでは、Eclipse-> Preferences-> JavaFXにします。シーンビルダホームのパスをします。






Scene Builderプロジェクトは、JavaFX by Oracleをしてされ、OpenJFXプロジェクトの

オープンソースです。

Oracle は 、Java SE 8u40のリリースにJavaFXのみをむScene Builder v 2.0までバイナリをしていたので、 `Spinner`コントロールのようなはまれていません。

Gluonはバイナリリリースのディストリビューションをきぎ、のScene Builder 8+をすべてのプラットフォームでここからダウンロードできます 。

これには、JavaFXのの、のとバグがまれています。

、リクエスト、プルリクエストをできるオープンソースプロジェクトがここにあります。

Oracleレガシー·バイナリは、 ここからダウンロードできます 。

## チュートリアル

シーンビルダのチュートリアルはここにあります

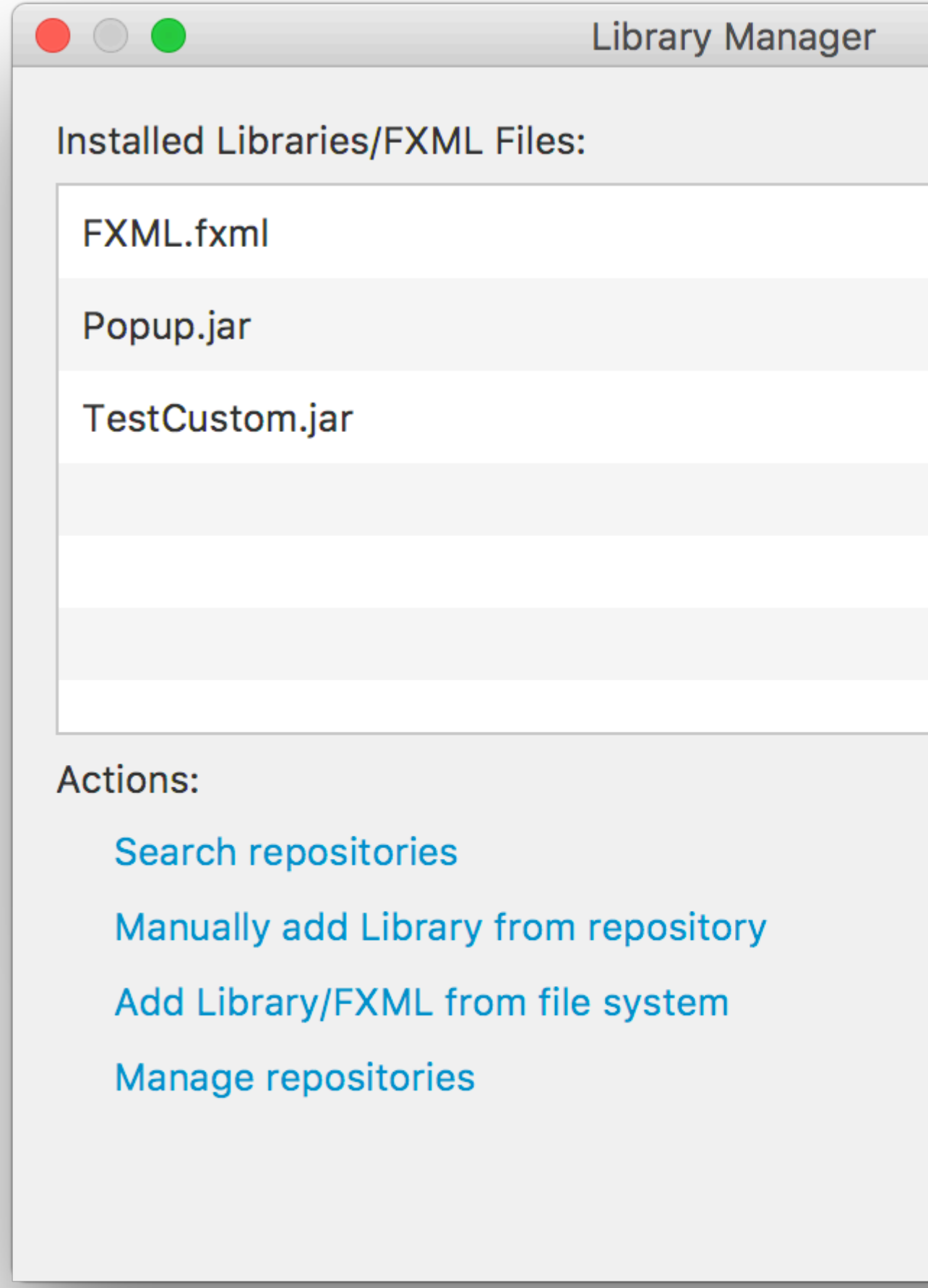
- Oracle Scene Builder 2.0 チュートリアル

FXMLチュートリアルはここでつけることができます。

- Oracle FXML チュートリアル

## カスタムコントロール

Gluonは、ライブラリマネージャScene Builder 8.2.0でをして、サードパーティのjarファイルをカスタムコントロールでインポートすることをにするしいをにしました。


Library Manager
Installed Libraries/FXML Files:
FXML.fxml
Popup.jar
TestCustom.jar
Actions:
Search repositories
Manually add Library from repository
Add Library/FXML from file system
Manage repositories

するなプロジェクトです。それは3つのファイルだけをんでいます

メインアプリケーションクラス

```
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);
    }
}
```

*FXML*ファイル

```
<?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>
```

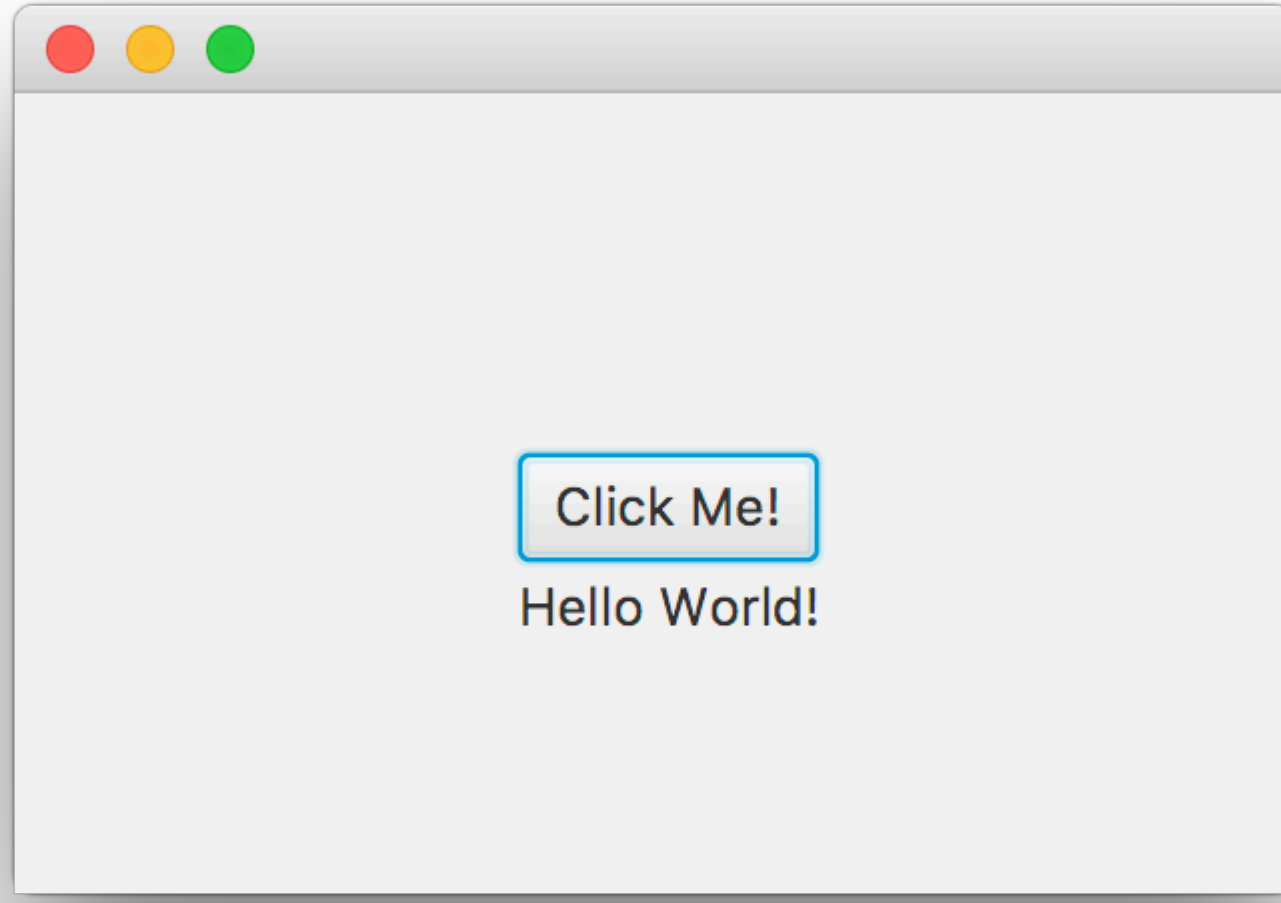
コントローラ

```
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!");
    }

}
```

る

プロジェクトのビルドとには、クリックなボタンきのさなウィンドウがされます。




い

には、メインアプリケーションクラスで、FXMLLoaderがロードされ`basicFXML.fxml`によってされるように、JAR /クラスパスから`FXMLLoader.load(getClass().getResource("BasicFXML.fxml"))`

`basicFXML.fxml`ロードすると、 `basicFXML.fxml` `fx:controller="org.stackoverflow.BasicFXMLController"`でされているように、ローダーはコントローラクラスのをつけます。

に、ローダーはそのクラスのインスタンスをし、 `fx:id`をつすべてのオブジェクトをFXMLにしようとし、コントローラクラスの`@FXML`アノテーションでマークされます。

このサンプルでは、FXMLLoaderは`<Label ... fx:id="label"/>`についてラベルをし、ラベルインスタンスを`@FXML private Label label;`。

に、FXMLがロードされると、FXMLLoaderはコントローラの`initialize`メソッドをびし、アクションハンドラをボタンにするコードがされます。

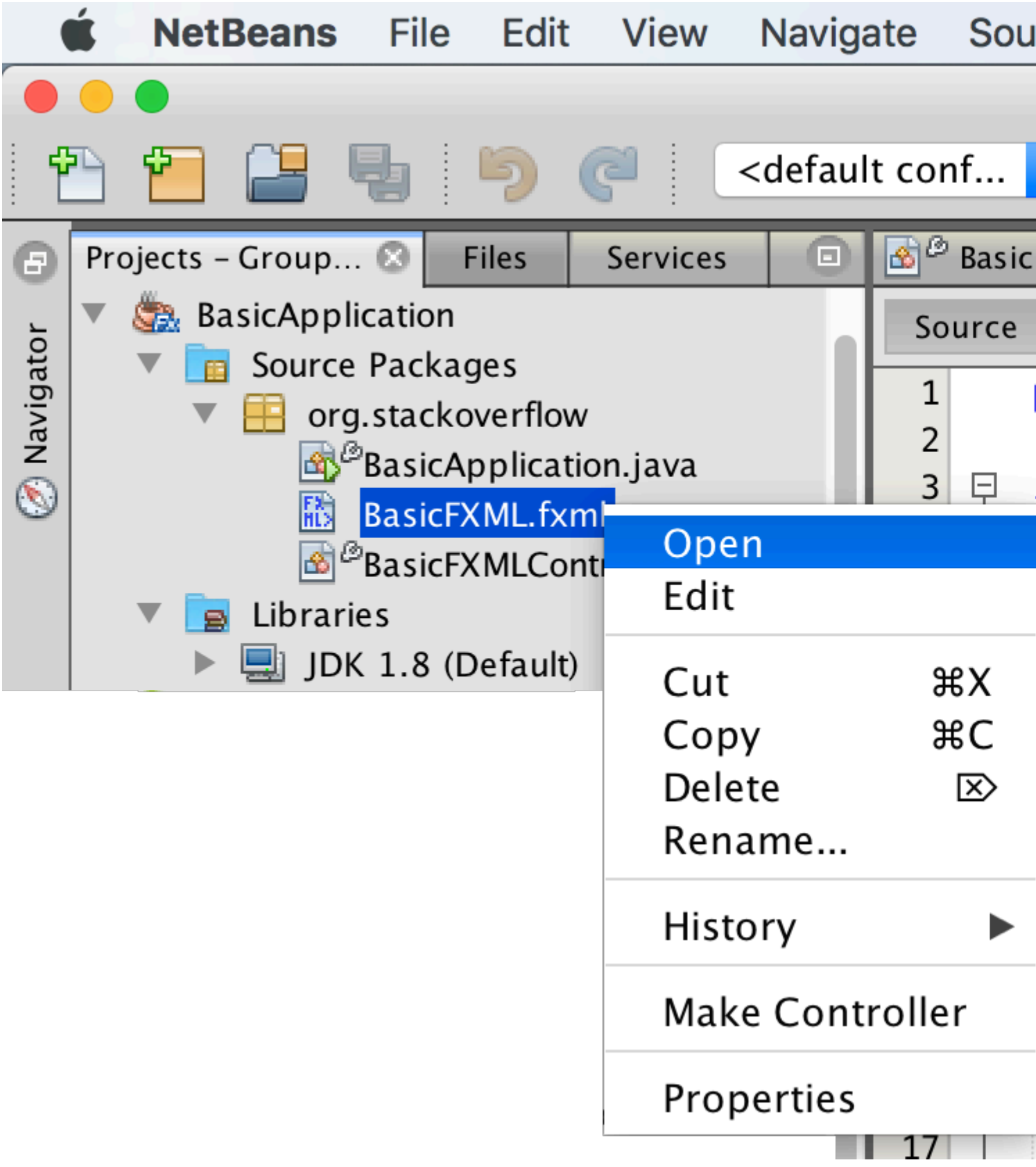
FXMLファイルはIDEでできますが、IDEはなチェックとをしますが、なガイダンスはしていないため、おめできません。

のは、Scene BuilderでFXMLファイルをき、すべてのをファイルにすることです。

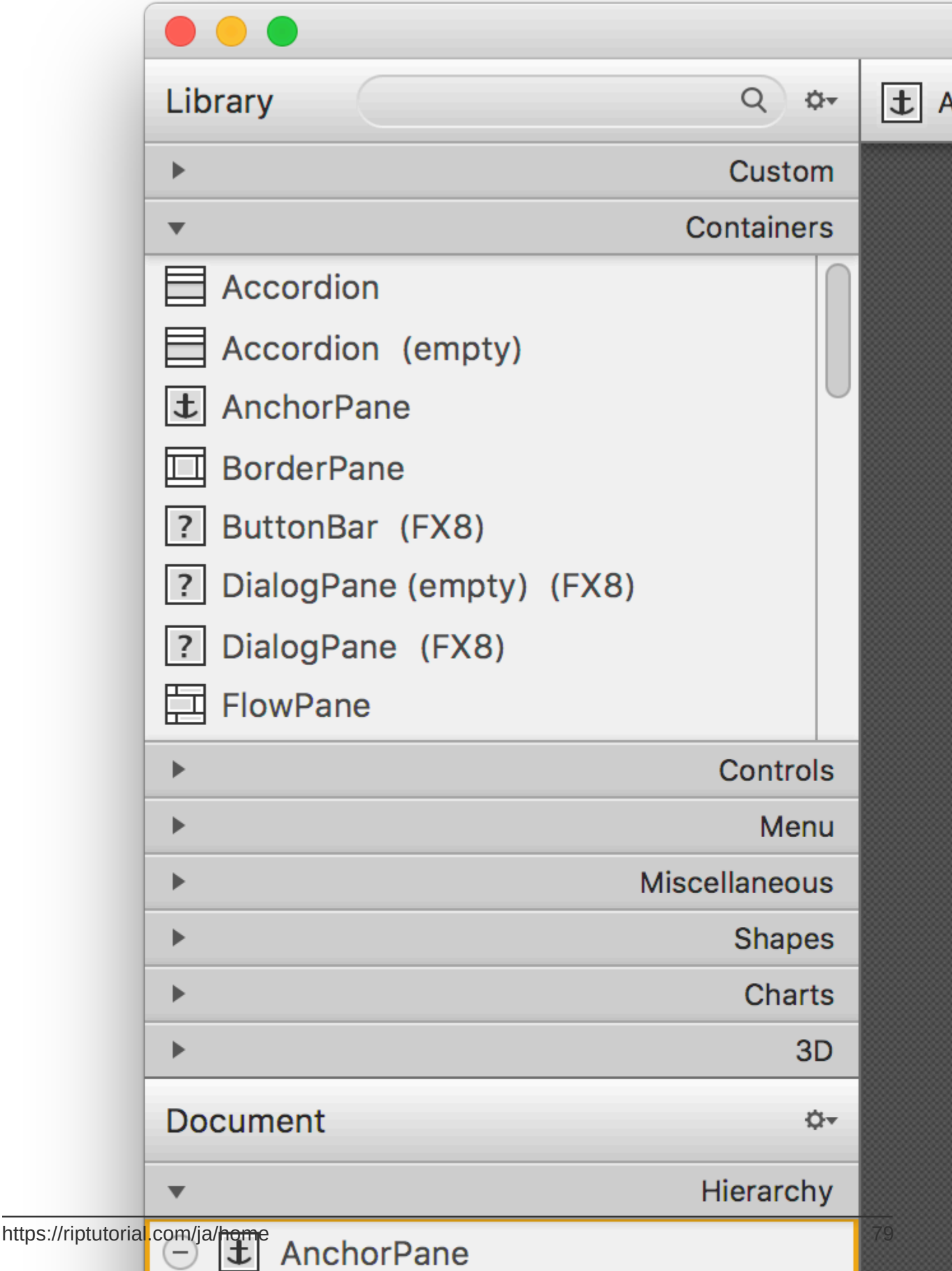
シーンビルダをしてファイルをくことができます

または、IDEからScene Builderでファイルをくこともできます

- NetBeansの[プロジェクト]タブで、ファイルをダブルクリックするか、クリックして[ `Open` ]をします。
- IntelliJのプロジェクトタブで、ファイルをクリックし、 `Open In Scene Builder`をします。
- Eclipseのプロジェクトタブで、ファイルをクリックし、「 `Open with Scene Builder`します。

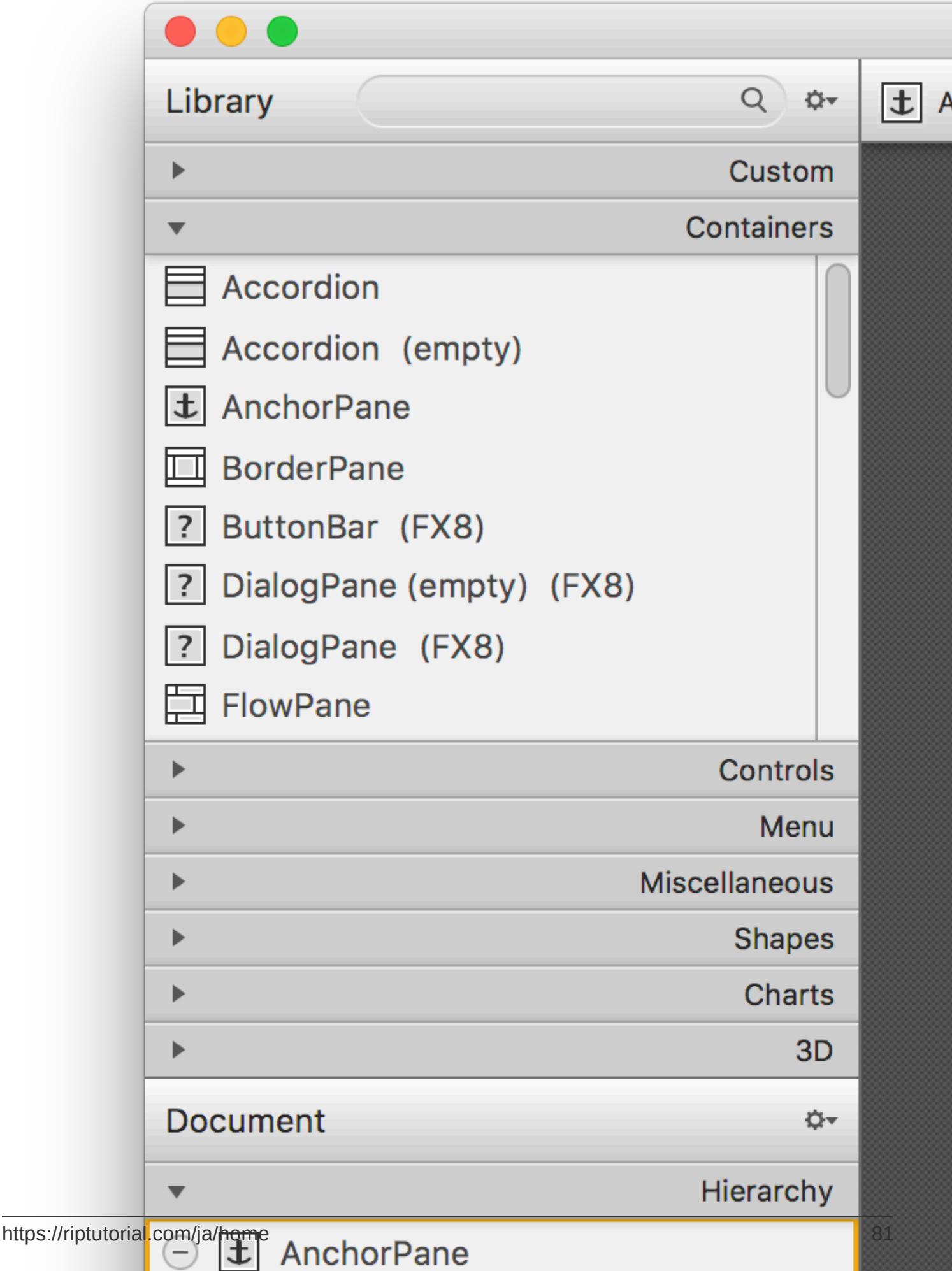



シーンビルダがしくインストールされ、そのパスがIDEにされているの「」を、ファイルがきます


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

プロパティとレイアウトのはのペインでできます。

FXMLをJavaコードにできるidタグの1つは`fx:id`です。 `Code`ペインですることができます


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

、ファイルをするときにシーンビルダでそのファイルがされます。

オンラインでシーンビルダーをむ https://riptutorial.com/ja/javafx/topic/5445/シーンビルダー

# 13: スレッディング

## Examples

### Platform.runLaterをしたUIの

のは、JavaFXアプリケーションスレッドではしないでください。これにより、JavaFXがUIをしなくなり、UIがフリーズすることがなくなります。

さらに、「ライブ」シーングラフのである`Node`へのは、JavaFXアプリケーションスレッドでするがあります。 `Platform.runLater`をすると、JavaFXアプリケーションスレッドでこれらのをできます。

のは、のスレッドから`Text Node`りしするをしています。

```
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);
    }

}
```

## UIアップデートのグループ

のコードでは、ボタンクリック、UIがしなくなります。これは、 `Platform.runLater`びしがすぎるためです。 ボタンをクリックしたに`ListView`スクロールしてみてください。

```
@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();
}
```

のをするのではなく、これをぐために、のコードでは、 `AnimationTimer`をしてフレームごとにを1だけします。

```
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();
        }
    }

}
```

`Updater`クラスをしてプログラムをグループすることができます。

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

...

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

## JavaFX Serviceのい

`JavaFX Thread`なタスクをするわりに、 `Service`するがあります 。 にサービスとはですか

サービスは、しい`Thread`をするたびにしい`Thread`し、それにタスクをしてらかのをうクラスです。サービスはをすことも、をすこともできません。

は、いくつかのをい、 `Map<String,String>(` をすJavaFX Serviceのなです。

```
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;
```

```
                }
            };
        }

}
```

オンラインでスレッディングをむ https://riptutorial.com/ja/javafx/topic/2230/スレッディング

# 14: ダイアログ

JavaFX 8アップデート40でダイアログがされました。

## Examples

### TextInputDialog

`TextInputDialog`すると、ユーザーにの`String`をするようにできます。

```
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`すると、オプションのリストから1つのをできます。

```
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);
```

### アラート

`Alert`は、のボタンをし、ユーザーがクリックしたボタンにじてをするなポップアップです。

これにより、ユーザーがにプライマリステージをじることをむかどうかをできます。

```
@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();
}
```

ボタンのテキストは、 `Locale`によってにされることにしてください。

### カスタムボタンのテキスト

ボタンにされるテキストは、 `ButtonType`インスタンスをですることでカスタマイズできます。

```
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();
```

オンラインでダイアログをむ https://riptutorial.com/ja/javafx/topic/3681/ダイアログ

# 15: チャート

## Examples

グラフ

PieChartクラスは、スライスにされたのでデータをします。すべてのスライスは、のにするパーツをします。グラフデータは、 PieChart.Dataオブジェクトにラップされます。 PieChart.Dataオブジェクトには、パイスライスのとするの2つのフィールドがあります。

## コンストラクタ

グラフをするには、 PieChartクラスのオブジェクトをするがあります。 2つのコンストラクタがされます。そのうちの1つは、データがsetDataメソッドでされていないり、もされないのグラフをします。

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

もう1つはPieChart.Data ObservableListをパラメータとしてすがあります。

```
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
```

## データ

スライスサイズはすべてののにしてされるため、パイスライスのはずしも100になるはありません。

データエントリがリストにされるによって、チャートのがされます。デフォルトではりにされますが、このはになります。

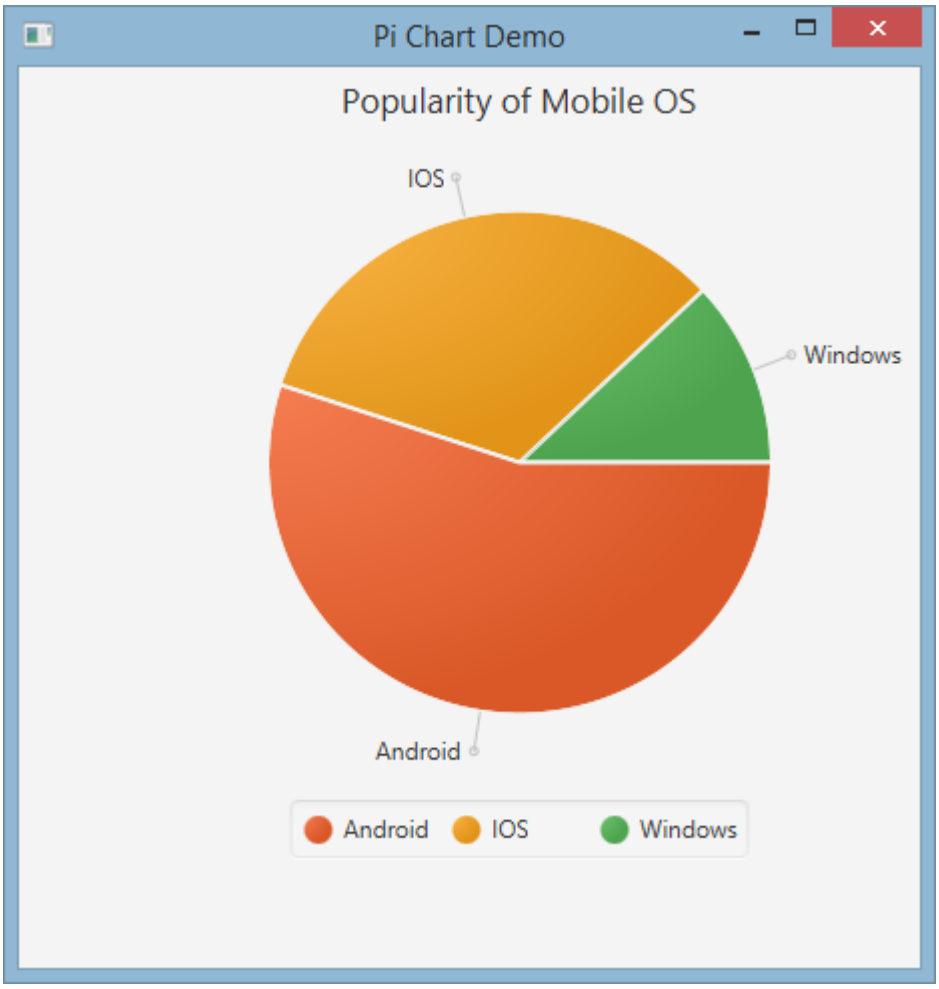
```
pieChart.setClockwise(false);
```

---

のでは、なグラフをします。

```
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);
    }
}
```




# グラフ

デフォルトでは、 `PieChart`はイベントをしませんが、スライスがJavaFX `Node`であるため、このはできます。

のでは、データをしてグラフにりてた、はされているをユーザーにできるように、スライスにツールチップをしてデータセットをします。

```
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);
});
```

れグラフ

`LineChart`クラスは、データをでされたのデータポイントとしてします。データポイントは`XYChart.Data`オブジェクトにラップされ、データポイントは`XYChart.Series`グループされ`XYChart.Series`。

`XYChart.Data`オブジェクトには、チャートのxとyのにする`getXValue`と`getYValue`をしてアクセスできる2つのフィールドがあります。

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

---

`LineChart`をするに、そのをするがあります。たとえば、 `NumberAxis`クラスのデフォルトでのないコンストラクタは、ができており、これするのないレンジングをします。

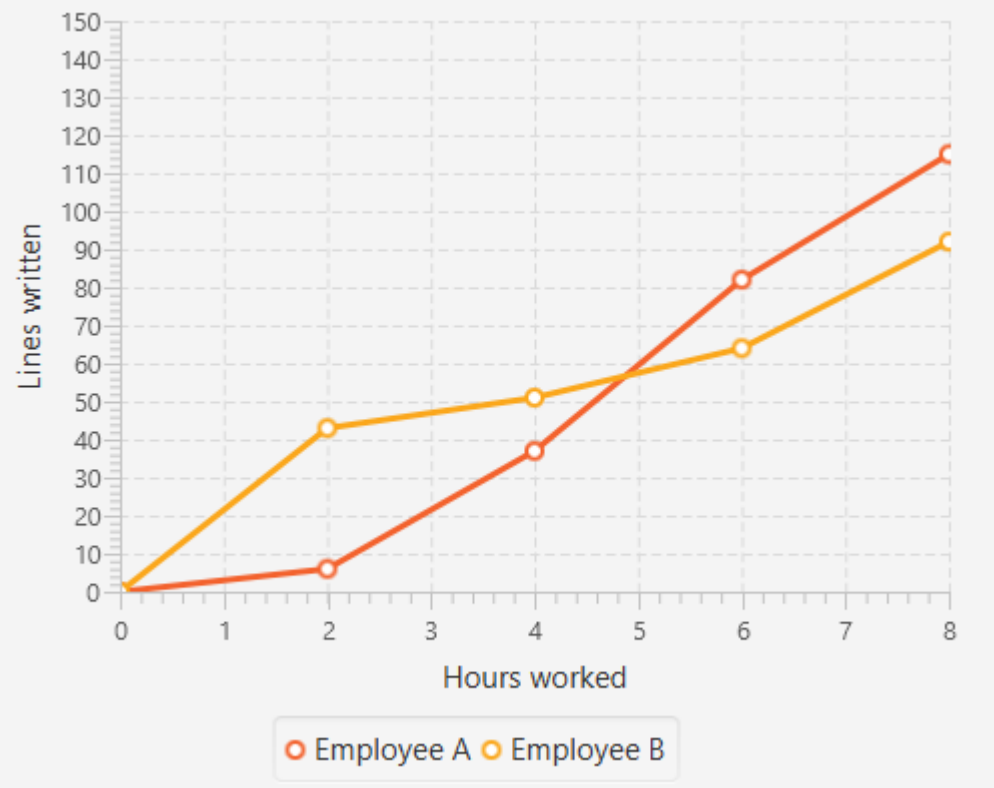
```
Axis xAxis = new NumberAxis();
```

---

のなでは、じグラフにされる2つののデータをします。のラベル、、ティックはにされています。

```
@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);
}
```




オンラインでチャートをむ https://riptutorial.com/ja/javafx/topic/2631/チャート

# 16: プロパティと

プロパティはであり、リスナをできます。これらは`Node`のプロパティ—にしてされます。

## Examples

### プロパティとの

### プロパティ

プロパティのタイプによっては、1つのプロパティにして3つのメソッドがあります。 `<property>`は`<property>`のをし、 `<Property>`はののをつプロパティのをします。そして、 `T`プロパティのとする。プリミティブラッパ—のために、たちはここにプリミティブ、えば、`String`のための`StringProperty`と`double`ため`ReadOnlyDoubleProperty`。

| メソッド | パラメ—ター | りの | |
|---|---|---|---|
| `<property>Property` | `()` | プロパティそのもの、えば `DoubleProperty`、 `ReadOnlyStringProperty`、 `ObjectProperty<VPos>` | リスナ—/バインディングをするためのプロパティをします。 |
| `get<Property>` | `()` | `T` | プロパティにラップされたをす |
| `set<Property>` | `(T)` | `void` | プロパティのをする |

みりのプロパティの、セッタ—はしないことにしてください。

### みりリストのプロパティ

みりリストのプロパティは、ゲッタ—メソッドのみをするプロパティです。このようなプロパティのは`ObservableList`、ましくはがされています。このプロパティのはされません。わりに`ObservableList`のをすることができます。

### のプロパティをみげる

みりリストプロパティとに、みりマッププロパティはゲッタ—をし、プロパティのわりにコンテンツをすることができます。 getterは`ObservableMap`します。

### StringPropertyの

のでは、プロパティのし`StringProperty`こののおよびするをし`ChangeListener`これに。

```
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");
    }

}
```

## ReadOnlyIntegerPropertyの

このは、みりのラッパー･プロパティーをして、きめないプロパティーをするをしています。この、 `cost`と`price`はできますが、 `profit`はに`price - cost`となり`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);
    }

}
```

オンラインでプロパティとをむ https://riptutorial.com/ja/javafx/topic/4436/プロパティと

# 17: ページネーション

## Examples

### ページネーションをする

JavaFXのページネーションは、コールバックをしてアニメーションでされるページをします。

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

これは、のボタンのリスト0..ゼロコンストラクタはのページをするので。 `setPageFactory`は、intをとるコールバックをとり、そのインデックスになノードをします。

```
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();
```

これは5ごとにページをめます。

い

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

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

`fiveSecondsWonder`は、サイクルがするたびにイベントをさせるタイムラインです。この、サイクルタイムは5です。

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

ページネーションにチェックをれてください。

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

にするようにタイムラインをします。

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

のページをする

```
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)));
```

パスはファイルシステムのパスではなく、URLでなければならないことにしてください。

い

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

のすべてはちょうどです、これはのがこっているところです。 `setPageFactory`は、intをとるコールバックをとり、そのインデックスになノードをします。のページはリストののにマッピングされ、2のはリストの2のにマッピングされます。

オンラインでページネーションをむ https://riptutorial.com/ja/javafx/topic/5165/ページネーション

# 18: ボタン

## Examples

### アクションリスナーの

ボタンがアクティブになるとアクションイベントがしますたとえば、クリック、ボタンのキーバインドがされたなど。

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

Java 8+をしているは、アクションリスナーにlambdasをできます。

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

### ボタンにグラフィックをする

ボタンはグラフィックをつことができます。 `graphic`は、 `ProgressBar`ようなのJavaFXノードにすることができます

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

`ImageView`

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

またはのボタン

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

### ボタンをする

`Button`はです

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

これにより、にテキストやグラフィックをまないしい`Button`がされます。

テキストきのButtonをするは、パラメータとしてStringをとるコンストラクタをします Button textPropertyをします。

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

グラフィックをまたはのNodeにつButtonをするは、のコンストラクタをします。

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

## デフォルトおよびキャンセルボタン

Button APIをすると、 Sceneりてられたアクセラレータのリストにアクセスしたり、キーイベントをにしたりするなく、のキーボードショートカットをボタンににりてることができます。つまり、 setDefaultButtonとsetCancelButton 2つのなメソッドがされています。

- setDefaultButtonをtrueすると、 KeyCode.ENTERイベントをけるたびにButtonがKeyCode.ENTERます。
- setCancelButtonをtrueすると、 KeyCode.ESCAPEイベントをけるたびにButtonがsetCancelButtonしtrue。

のでは、またはエスケープキーがされたときに、フォーカスされているかどうかになくされる2つのボタンをつSceneをします。

```
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);
```

これらのKeyEventsがNodeによってされた、のコードはしません

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

オンラインでボタンをむ https://riptutorial.com/ja/javafx/topic/5162/ボタン

# 19: ラジオボタン

## Examples

### ラジオボタンの

ラジオボタンをすると、ユーザーはえられたの1つのをできます。 `RadioButton`は、それのテキストをするが2つあります。デフォルトのコンストラクタである`RadioButton()`をし、 `setText(String)`メソッドでテキストをするか、のコンストラクタ`RadioButton(String)`をしてします。

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

`RadioButton`は`Labeled`であるため、 `RadioButton`された`Image`することもでき`RadioButton`。 コンストラクタの1つで`RadioButton`をしたは、 `setGraphic(ImageView)`ように`setGraphic(ImageView)`メソッドを`setGraphic(ImageView)`て`Image`をします。

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

### ラジオボタンでグループをする

`ToggleGroup`は`RadioButton`をするためにされ、グループの1つだけをできます。

のようなな`ToggleGroup`します。

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

`Togglegroup`をした、 `Togglegroup` `setToggleGroup(ToggleGroup)`をして`RadioButton`りてることができます。 `setSelected(Boolean)`をして、 `RadioButton`の1つをにします。

```
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);
```

### ラジオボタンのイベント

、 `ToggleGroup` `RadioButton`の1つがされると、アプリケーションはアクションをします。は、

`setUserData(Object)`さ`RadioButton`たした`RadioButton`ユーザデータをするです。

```
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());
    }
});
```

## ラジオボタンのフォーカスをする

3つのうち2つの`RadioButton`が`setSelected(Boolean)`であらかじめされているとします。デフォルトではフォーカスはの`RadioButton`ります。 これをするには、 `requestFocus()`メソッドをします。

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

オンラインでラジオボタンをむ https://riptutorial.com/ja/javafx/topic/5906/ラジオボタン

# 20: レイアウト

## Examples

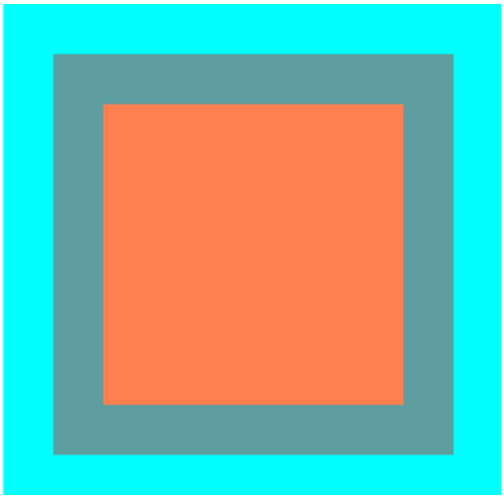
### StackPane

`StackPane`は、わせのスタックにをレイアウトします。

のzオーダーは、リスト `getChildren`びしでアクセスのでされます`getChildren`のはスタックのにあり、のはスタックのにあります。

stackpaneは、のサイズをして、のコンテンツをめようとします。 `StackPane`のをたすためにのサイズをできないサイズできなかったかサイズができなかったため、スタックペインの`alignmentProperty`をしてにえられます`Pos.CENTER`.デフォルトは`Pos.CENTER`.

```
// 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およびVBox

`HBox`と`VBox`レイアウトはによくていて、ともを1にレイアウトしています。

の

HBoxまたはVBoxボーダーおよび/またはパディングセットがある、そのインセットにコンテンツがVBoxされます。

どものにえるにかかわらず、マネージド・チャイルドをします。されていないはされます。

コンテンツのえはalignmentプロパティによってされます。デフォルトはPos.TOP_LEFTです。

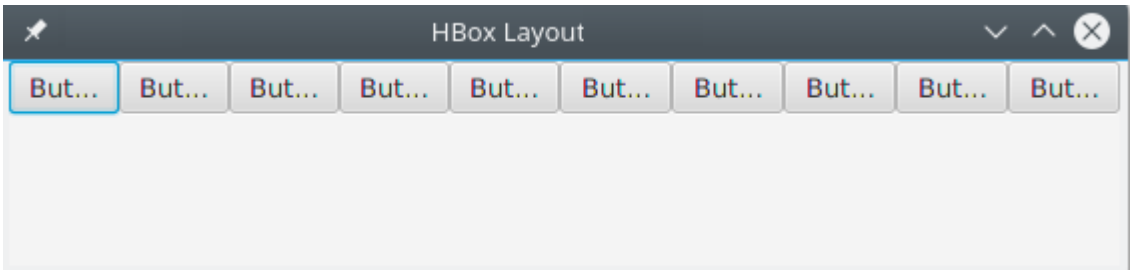
### HBox

HBoxは、からへ1に1つのをします。

HBoxはサイズな をの sにリサイズし、そのheightプロパティのさをしてさをするか、さをfillHeightのデフォルトはtrueにするかをします。

HBoxの

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



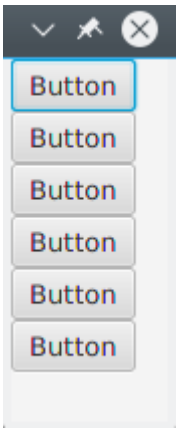

### VBox

VBoxは、そのをからににします。

VBoxは、ウィンドウのサイズをみのさにし、fillWidthプロパティをしてをリサイズしてをするか、をどおりにするかをしますfillWidthのデフォルトはtrue。

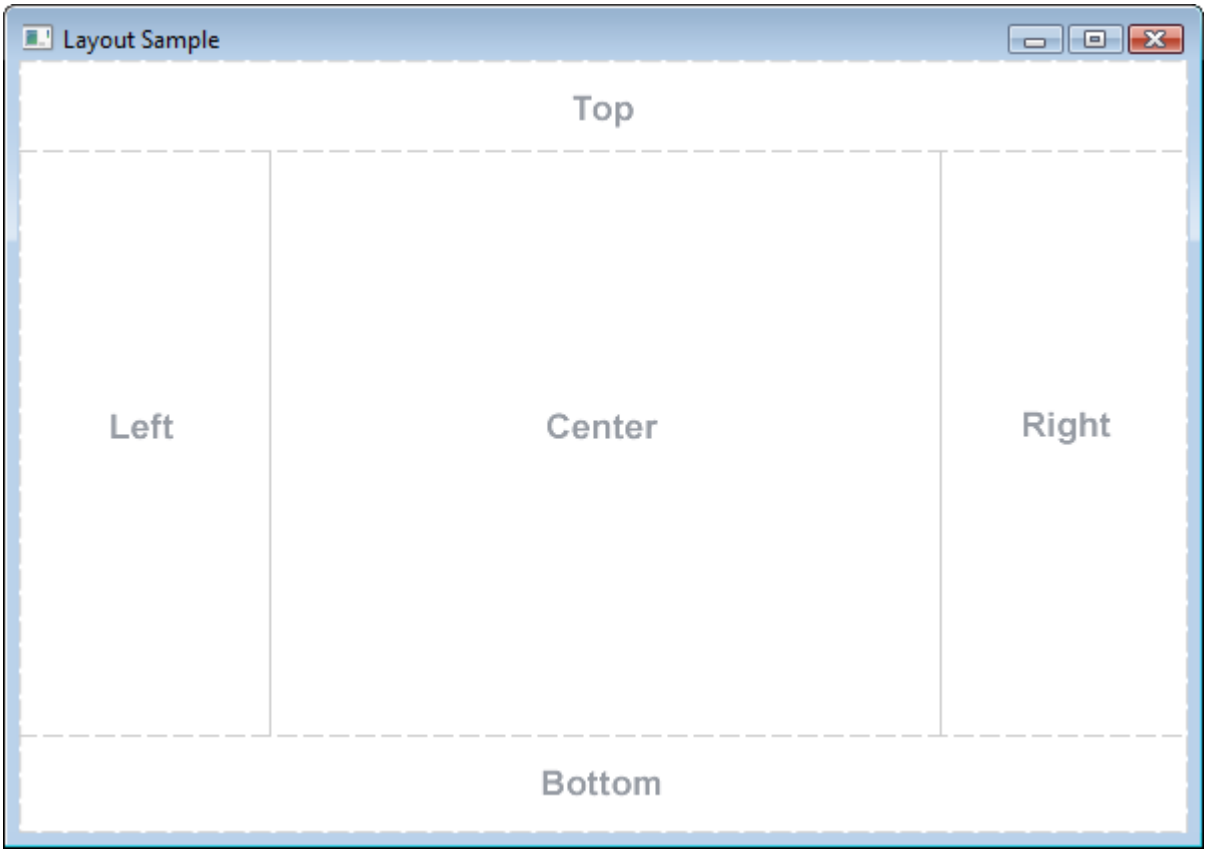
VBoxの

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




## BorderPane

`BorderPane`は5つのなるにかれています。




ボーダー `Top` 、 `Right` 、 `Bottom` 、 `Left` は、そのについてサイズがされています。デフォルトでは、なものだけをり、 `Center`エリアはりのスペースをとります。ボーダーエリアがの、スペースをらない。

にまれるは1つだけです。これは、 `setTop(Node)` 、 `setRight(Node)` 、 `setBottom(Node)` 、 `setLeft(Node)` 、 `setCenter(Node)`のメソッドをしてできます。のレイアウトをして、のを1つのにすることができます。

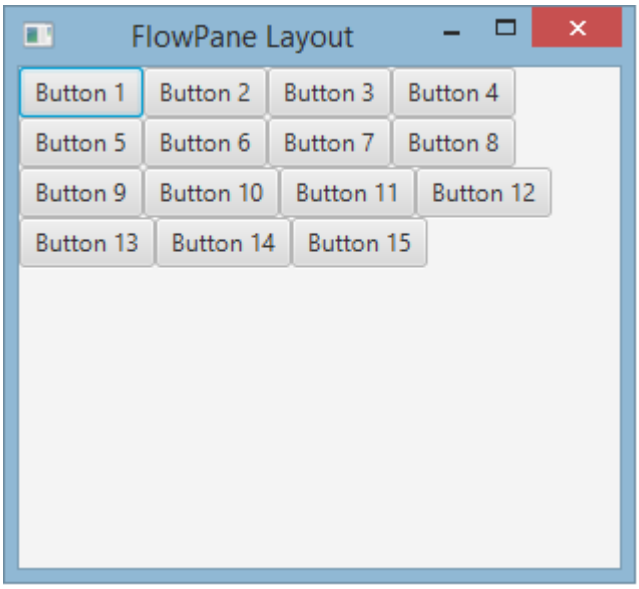
```
//BorderPane example
BorderPane pane = new BorderPane();
```

```
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は、なまたはスペースについて、ノードをまたはにレイアウトします。スペースがすべてのノードののよりもさい、ノードはのにラップされます。スペースがすべてのノードのさのよりもさい、ノードをのにラップします。のは、デフォルトのレイアウトをしています。

```
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);
    }
}
```




デフォルトの`FlowPane`コンストラクタ

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

の`FlowPane`コンストラクタ

```
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.
```

レイアウトにノードをすると、`Pane`の`add()`または`addAll()`メソッドがされます。

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

デフォルトでは、`FlowPane`はノードをからにレイアウトします。フローのをするには、`Pos`のをして`setAlignment()`メソッドをびします。

にされるフローアライメント

```
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は、そのをとのなグリッドにレイアウトします。

# GridPaneの

はGridPaneののにくことができ、の/にまたがることができますデフォルトスパンは1。グリッドのはレイアウトによってされます。

| | |
|---|---|
| columnIndex | のレイアウトがされる。 |
| rowIndex | のレイアウトがまる。 |
| columnSpan | のレイアウトがにまたがるの |
| rowSpan | のレイアウトがにまたがるの |

グリッドパネルはコンテンツをめるためにグリッドをに/するため、/のをするはありません。

## GridPaneにをする

しいNodeをGridPaneにするには、 GridPaneクラスのメソッドをしてのレイアウトをするがありGridPaneこれらのをGridPaneインスタンスにできます。

```
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は、これらのステップをみわせるなをします

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

GridPaneクラスは、なセッター—メソッドをして、のとのスパンをします。

```
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
```

# とのサイズ

デフォルトでは、とのサイズはコンテンツにわせてされます。 とのサイズをにするがある、RowConstraintsインスタンスとColumnConstraintsインスタンスをRowConstraintsにできGridPane。 これらの2つのをすると、ののサイズをして、のを100ピクセル、2を200ピクセルにします。

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

デフォルトではGridPane gridpaneは、そのましいサイズよりもきなサイズがされたでもみのきさに/のサイズをします。 /サイズをサポートするために、のcontstaintsクラスには、サイズ、サイズ、およびサイズの3つのプロパティがされています。

さらに、 ColumnConstraintsはsetHGrowをし、 RowConstraintsは、およびのにをえる setVGrowメソッドをします。にされた3つのはのとおりです。

- に にまたはし、にまたはしているのレイアウトとのスペースのまたはをしようとする
- **Priority.SOMETIMES** またはがににされているレイアウトがない、またはそれらのレイアウトがまたはしたのすべてをしなかった、のまたはをしますのレイアウトエリアのものがあります。
- **Priority.NEVER** レイアウトは、でなのまたはがあるには、またはしません。

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

でしたのサイズは100ピクセルで、300ピクセルのにするまでにしようとします。

とのサイズのサイジングをすることもできます。のでは、のがグリッドパネルのの40をめ、2のが60をめるGridPaneをしGridPane。

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

## グリッドセルのの

アラインメントNode SをいてすることができるsetHalignmentのメソッドColumnConstraintsクラスとsetValignmentのメソッドRowConstraintsクラス。

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

RowConstraints row1 = new RowConstraints();
```

```
row1.setValignment(VPos.CENTER);
```

## TilePane

タイルペインのレイアウトは**FlowPane**レイアウトにています。 TilePaneは、セルまたはタイルがじサイズのグリッドにすべてのノードをします。これは、ノードをまたはのきれいなとにさせます。

```
    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);
        }
    }
```


タイルパネルをするには

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

setHgapsetVgapメソッドは、とのにギャップをるためにされます。レイアウトのをのようにすることもできます。

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

## AnchorPane

`AnchorPane` aは、コンテンツをそのからのにくことをにするレイアウトです。

`AnchorPane`は4つのとをるがあります。これらのメソッドののパラメータは、`Node`です。セッターの2のパラメーターは、する`Double`です。このは、されたのがないことをす`null`ことができます。

| セッターメソッド | ゲッターメソッド |
|---|---|
| setBottomAnchor | getBottomAnchor |
| setLeftAnchor | getLeftAnchor |
| setRightAnchor | getRightAnchor |
| setTopAnchor | getTopAnchor |

のでは、からされたにノードをします。

`center`も、されたをからするようにサイズがされます。ウィンドウのサイズがされたときのをし

ます。

```
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();
}
```

オンラインでレイアウトをむ https://riptutorial.com/ja/javafx/topic/2121/レイアウト

# 21:

## Examples

な

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

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

これは、ダイアログボックスをせずに、デフォルトのプリンタにします。デフォルトのプリンタをするには、 `PrinterJob#createPrinterJob(Printer)` をしてのプリンタをします。これをして、システムのすべてのプリンタをできます。

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

システムダイアログでの

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

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

オンラインでをむ https://riptutorial.com/ja/javafx/topic/5157/

# クレジット

| S. No | | Contributors |
|---|---|---|
| 1 | javafxをいめる | Community, CraftedCart, D3181, DVarga, fabian, Ganesh, Hendrik Ebbers, Petter Friberg |
| 2 | CSS | fabian |
| 3 | FXMLとコントローラ | D3181, fabian, James_D |
| 4 | JavaFXでの | ItachiUchiha, Joffrey, Nico T, P.J.Meisch |
| 5 | JavaFXバインディング | Alexiy |
| 6 | ScrollPane | Bo Halim |
| 7 | TableView | Bo Halim, fabian, GltknBtn |
| 8 | WebViewとWebEngine | fabian, J Atkin, James_D, P.J.Meisch, Squidward |
| 9 | Windows | fabian, GltknBtn |
| 10 | アニメーション | fabian, J Atkin |
| 11 | キャンバス | Dth |
| 12 | シーンビルダー | Ashlyn Campbell, José Pereda |
| 13 | スレッディング | Brendan, fabian, GOXR3PLUS, James_D, Koko Essam, sazzy4o |
| 14 | ダイアログ | fabian, GltknBtn, Modus Tollens |
| 15 | チャート | Dth, James_D, Jinu P C |
| 16 | プロパティと | fabian |
| 17 | ページネーション | fabian, J Atkin |
| 18 | ボタン | Dth, DVarga, J Atkin, Maverick283, Nico T, Squidward |
| 19 | ラジオボタン | Nico T |

| | | |
|---|---|---|
| 20 | レイアウト | DVarga, fabian, Filip Smola, Jinu P C, Sohan Chowdhury, trashgod |
| 21 | | J Atkin, Squidward |