



EBook Gratuito

APPENDIMENTO

javafx

Free unaffiliated eBook created from
Stack Overflow contributors.

#javafx

Sommario

| | |
|---|-----------|
| Di..... | 1 |
| Capitolo 1: Iniziare con javafx..... | 2 |
| Osservazioni..... | 2 |
| Versioni..... | 2 |
| Examples..... | 2 |
| Installazione o configurazione..... | 2 |
| Ciao programma mondiale..... | 3 |
| Capitolo 2: Animazione..... | 5 |
| Examples..... | 5 |
| Animazione di una proprietà con la cronologia..... | 5 |
| Capitolo 3: Binding JavaFX..... | 6 |
| Examples..... | 6 |
| Legame semplice della proprietà..... | 6 |
| Capitolo 4: CSS..... | 7 |
| Sintassi..... | 7 |
| Examples..... | 7 |
| Usando i CSS per lo styling..... | 7 |
| Estendere il rettangolo aggiungendo nuove proprietà stylable..... | 9 |
| Nodo personalizzato..... | 9 |
| Capitolo 5: finestre..... | 13 |
| Examples..... | 13 |
| Creare una nuova finestra..... | 13 |
| Creazione di finestre di dialogo personalizzate..... | 13 |
| Creazione di finestre di dialogo personalizzate..... | 18 |
| Capitolo 6: Finestre di dialogo..... | 25 |
| Osservazioni..... | 25 |
| Examples..... | 25 |
| TextInputDialog..... | 25 |
| ChoiceDialog..... | 25 |
| Mettere in guardia..... | 26 |

| | |
|--|-----------|
| Esempio..... | 26 |
| Testo del pulsante personalizzato..... | 26 |
| Capitolo 7: FXML e controller..... | 27 |
| Sintassi..... | 27 |
| Examples..... | 27 |
| Esempio FXML..... | 27 |
| Controller annidati..... | 29 |
| Definisci i blocchi e..... | 31 |
| Trasmissione dei dati a FXML - accesso al controller esistente..... | 32 |
| Trasmissione dei dati a FXML - Specifica dell'istanza del controllore..... | 33 |
| Passaggio dei parametri a FXML - utilizzando un controllerFactory..... | 34 |
| Creazione di istanze in FXML..... | 36 |
| Una nota sulle importazioni..... | 37 |
| @NamedArg annotato @NamedArg..... | 37 |
| Nessun costruttore di args..... | 38 |
| fx:value attributo fx:value..... | 38 |
| fx:factory..... | 38 |
| <fx:copy>..... | 39 |
| fx:constant..... | 39 |
| Impostazione delle proprietà..... | 39 |
| tag <property>..... | 39 |
| Proprietà di default..... | 39 |
| property="value" attributo property="value"..... | 40 |
| setter statici..... | 40 |
| Tipo coercizione..... | 40 |
| Esempio..... | 41 |
| Capitolo 8: Grafico..... | 44 |
| Examples..... | 44 |
| Grafico a torta..... | 44 |
| Costruttori..... | 44 |
| Dati..... | 44 |

| | |
|---|-----------|
| Esempio | 44 |
| Produzione: | 45 |
| Grafico a torta interattivo | 46 |
| Grafico a linee..... | 46 |
| assi | 47 |
| Esempio | 47 |
| Produzione: | 48 |
| Capitolo 9: Internazionalizzazione in JavaFX | 49 |
| Examples..... | 49 |
| Caricamento del pacchetto di risorse..... | 49 |
| controllore..... | 49 |
| Cambio della lingua in modo dinamico quando l'applicazione è in esecuzione..... | 49 |
| Capitolo 10: layout | 55 |
| Examples..... | 55 |
| StackPane..... | 55 |
| HBox e VBox..... | 55 |
| BorderPane..... | 57 |
| FlowPane..... | 58 |
| GridPane..... | 60 |
| Figli della GridPane | 60 |
| Aggiunta di bambini a GridPane..... | 60 |
| Dimensione di colonne e righe | 61 |
| Allineamento di elementi all'interno delle celle della griglia..... | 62 |
| TilePane..... | 62 |
| AnchorPane..... | 63 |
| Capitolo 11: paginatura | 65 |
| Examples..... | 65 |
| Creazione di una paginazione..... | 65 |
| Avanzamento automatico..... | 65 |
| Come funziona..... | 65 |
| Crea una paginazione di immagini..... | 66 |

| | |
|--|-----------|
| Come funziona | 66 |
| Capitolo 12: Proprietà e osservabili | 67 |
| Osservazioni | 67 |
| Examples | 67 |
| Tipi di proprietà e denominazione | 67 |
| Proprietà standard | 67 |
| Visualizza le proprietà in sola lettura | 67 |
| Visualizza di sola lettura le proprietà | 67 |
| Esempio StringProperty | 68 |
| Esempio ReadOnlyIntegerProperty | 68 |
| Capitolo 13: Pulsante | 71 |
| Examples | 71 |
| Aggiunta di un listener di azioni | 71 |
| Aggiunta di un elemento grafico a un pulsante | 71 |
| Crea un pulsante | 71 |
| Pulsanti predefiniti e Annulla | 72 |
| Capitolo 14: Pulsante radio | 73 |
| Examples | 73 |
| Creazione di pulsanti radio | 73 |
| Utilizza i gruppi sui pulsanti di opzione | 73 |
| Eventi per i pulsanti radio | 73 |
| Richiesta di messa a fuoco per pulsanti di opzione | 74 |
| Capitolo 15: Scene Builder | 75 |
| introduzione | 75 |
| Osservazioni | 75 |
| Installazione di Scene Builder | 75 |
| Un po 'di storia | 78 |
| Esercitazioni | 79 |
| Controlli personalizzati | 79 |
| QUINDI domande | 80 |
| Examples | 80 |
| Progetto JavaFX di base tramite FXML | 80 |

| | |
|--|------------|
| Capitolo 16: ScrollPane | 86 |
| introduzione | 86 |
| Examples | 86 |
| A) Corrette le dimensioni del contenuto: | 86 |
| B) Dimensioni del contenuto dinamico: | 86 |
| Styling the ScrollPane: | 87 |
| Capitolo 17: Stampa | 88 |
| Examples | 88 |
| Stampa di base | 88 |
| Stampa con finestra di dialogo del sistema | 88 |
| Capitolo 18: TableView | 89 |
| Examples | 89 |
| Sample TableView con 2 colonne | 89 |
| PropertyValueFactory | 92 |
| Personalizzazione di TableCell in base all'elemento | 93 |
| Aggiungi pulsante a Tableview | 96 |
| Capitolo 19: Tela | 100 |
| introduzione | 100 |
| Examples | 100 |
| Forme di base | 100 |
| Capitolo 20: threading | 102 |
| Examples | 102 |
| Aggiornamento dell'interfaccia utente mediante Platform.runLater | 102 |
| Raggruppamento degli aggiornamenti dell'interfaccia utente | 103 |
| Come utilizzare il servizio JavaFX | 104 |
| Capitolo 21: WebView e WebEngine | 107 |
| Osservazioni | 107 |
| Examples | 107 |
| Caricamento di una pagina | 107 |
| Ottieni la cronologia delle pagine di una WebView | 107 |
| inviare avvisi JavaScript dalla pagina Web visualizzata al log delle applicazioni Java | 108 |
| Comunicazione tra app Java e Javascript nella pagina web | 108 |

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

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

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Capitolo 1: Iniziare con javafx

Osservazioni

JavaFX è una piattaforma software per la creazione e la distribuzione di applicazioni desktop, nonché applicazioni Internet ricche (RIA) che possono essere eseguite su un'ampia varietà di dispositivi. JavaFX è destinato a sostituire Swing come libreria GUI standard per Java SE.

L'IT consente agli sviluppatori di progettare, creare, testare, eseguire il debug e distribuire applicazioni rich client.

L'aspetto delle applicazioni JavaFX può essere personalizzato utilizzando Cascading Style Sheets (CSS) per lo stile (vedi [JavaFX: CSS](#)) e (F) I file XML possono essere utilizzati per strutturare strutture che facilitano la creazione o lo sviluppo di un'applicazione (vedere [FXML e controller](#)). Scene Builder è un editor visuale che consente la creazione di file fxml per un'interfaccia utente senza scrivere codice.

Versioni

| Versione | Data di rilascio |
|----------|------------------|
| JavaFX 2 | 2011-10-10 |
| JavaFX 8 | 2014/03/18 |

Examples

Installazione o configurazione

Le API JavaFX sono disponibili come funzionalità completamente integrata di Java SE Runtime Environment (JRE) e Java Development Kit (JDK). Poiché il JDK è disponibile per tutte le principali piattaforme desktop (Windows, Mac OS X e Linux), le applicazioni JavaFX compilate su JDK 7 e versioni successive vengono eseguite su tutte le principali piattaforme desktop. Il supporto per le piattaforme ARM è stato reso disponibile con JavaFX 8. JDK per ARM include i componenti base, grafici e di controllo di JavaFX.

Per installare JavaFX, installare la versione scelta dell'ambiente Java Runtime e del [kit di sviluppo Java](#).

Le funzionalità offerte da JavaFX includono:

1. API Java.
2. FXML e Scene Builder.
3. WebView.

4. Interoperabilità Swing.
5. Controlli dell'interfaccia utente e CSS incorporati.
6. Tema modenese
7. Funzionalità grafiche 3D.
8. API Canvas.
9. API di stampa.
10. Supporto RTF.
11. Supporto multitouch.
12. Supporto Hi-DPI.
13. Pipeline grafica con accelerazione hardware.
14. Motore multimediale ad alte prestazioni.
15. Modello di implementazione dell'applicazione autonomo.

Ciao programma mondiale

Il codice seguente crea una semplice interfaccia utente contenente un singolo `Button` che stampa una `String` sulla console al clic.

```
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);
}
}

```

La classe `Application` è il punto di ingresso di ogni applicazione JavaFX. È possibile avviare una sola `Application` e questo viene fatto utilizzando

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

Ciò crea un'istanza della classe `Application` passata come parametro e avvia la piattaforma JavaFX.

Quanto segue è importante per il programmatore qui:

1. Il primo `launch` crea una nuova istanza della classe `Application` (`HelloWorld` in questo caso). La classe `Application` pertanto necessita di un costruttore no-arg.
2. `init()` viene chiamato sull'istanza `Application` creata. In questo caso l'implementazione predefinita da `Application` non fa nulla.
3. viene chiamato `start` per l'istanza `Application` e lo `Stage` primario (= window) viene passato al metodo. Questo metodo viene automaticamente chiamato sul thread dell'applicazione JavaFX (thread della piattaforma).
4. L'applicazione viene eseguita fino a quando la piattaforma non determina il momento di spegnersi. Questo viene fatto quando l'ultima finestra è chiusa in questo caso.
5. Il metodo `stop` è invocato sull'istanza `Application`. In questo caso l'implementazione da `Application` non fa nulla. Questo metodo viene automaticamente chiamato sul thread dell'applicazione JavaFX (thread della piattaforma).

Nel metodo `start` viene costruito il grafico della scena. In questo caso contiene 2 `Node`: A `Button` e `StackPane`.

Il `Button` rappresenta un pulsante nell'interfaccia utente e `StackPane` è un contenitore per il `Button` che ne determina il posizionamento.

Viene creata una `Scene` per visualizzare questi `Node`. Infine, la `Scene` viene aggiunta allo `Stage` che è la finestra che mostra l'intera interfaccia utente.

Leggi Iniziare con javafx online: <https://riptutorial.com/it/javafx/topic/887/iniziare-con-javafx>

Capitolo 2: Animazione

Examples

Animazione di una proprietà con la cronologia

```
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();
```

Il modo più semplice e flessibile per utilizzare l'animazione in JavaFX è con la classe `Timeline`. Una timeline funziona utilizzando `KeyFrame`s come punti noti dell'animazione. In questo caso, sa che all'inizio (0 seconds) che `translateXProperty` deve essere zero e alla fine (2 seconds) che la proprietà deve essere 80 . Puoi anche fare altre cose come impostare l'animazione per invertire e quante volte dovrebbe essere eseguita.

Le linee temporali possono animare più proprietà contemporaneamente:

```
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
```

Questa animazione prende la proprietà `Y` da 0 (valore iniziale della proprietà) a 10 secondi in e termina a 90 in tre secondi. Nota che quando l'animazione ricomincia, `Y` torna a zero, anche se non è il primo valore nella timeline.

Leggi Animazione online: <https://riptutorial.com/it/javafx/topic/5166/animazione>

Capitolo 3: Binding JavaFX

Examples

Legame semplice della proprietà

JavaFX ha un'API di associazione, che fornisce metodi per legare una proprietà all'altra. Ciò significa che ogni volta che viene modificato il valore di una proprietà, il valore della proprietà associata viene aggiornato automaticamente. Un esempio di rilegatura semplice:

```
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
}
```

Puoi anche associare una proprietà primitiva con l'applicazione di un'addizione, sottrazione, divisione, ecc:

```
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'
}
```

Qualsiasi oggetto può essere inserito in SimpleObjectProperty:

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

È possibile creare binding bidirezionali. In questo caso, le proprietà dipendono l'una dall'altra.

```
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'
}
```

Leggi Binding JavaFX online: <https://riptutorial.com/it/javafx/topic/7014/binding-javafx>

Capitolo 4: CSS

Sintassi

- `Selettore NodeClass` / * per classe `Node` * /
- `.someclass` / * selettore per classe * /
- `#someId` / * selettore per id * /
- `[selettore1]> [selettore2]` / * selettore per un figlio diretto di un selettore di corrispondenza del nodo1 che corrisponde a `selector2` * /
- `[selettore1] [selettore2]` / * selettore per un discendente di un selettore di corrispondenza del nodo1 che corrisponde a `selettore2` * /

Examples

Usando i CSS per lo styling

I CSS possono essere applicati in più punti:

- `inline` (`Node.setStyle`)
- in un foglio di stile
 - a una `Scene`
 - come foglio di stile user-agent (non mostrato qui)
 - come foglio di stile "normale" per la `Scene`
 - a un `Node`

Ciò consente di modificare le proprietà styleable dei `Nodes` . Il seguente esempio dimostra questo:

Classe di applicazione

```
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;
}

```

Estendere il rettangolo aggiungendo nuove proprietà stylable

JavaFX 8

Nell'esempio seguente viene illustrato come aggiungere proprietà personalizzate che possono essere ridisegnate da CSS a un `Node` personalizzato.

Qui 2 `DoubleProperty` vengono aggiunti alla classe `Rectangle` per consentire l'impostazione della `width` e `height` da CSS.

Il seguente CSS potrebbe essere utilizzato per lo styling del nodo personalizzato:

```

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

```

Nodo personalizzato

```

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;  
}  
  
}
```

Leggi CSS online: <https://riptutorial.com/it/javafx/topic/1581/css>

Capitolo 5: finestre

Examples

Creare una nuova finestra

Per mostrare del contenuto in una nuova finestra, è necessario creare uno `Stage`. Dopo la creazione e l'inizializzazione `show` o `showAndWait` deve essere richiamato sull'oggetto `Stage`:

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

Parent content = root;

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

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

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

Nota: questo codice deve essere eseguito sul thread dell'applicazione JavaFX.

Creazione di finestre di dialogo personalizzate

È possibile creare finestre di dialogo personalizzate che contengono molti componenti ed eseguire molte funzionalità su di esso. Si comporta come un secondo stadio sul palco del proprietario. Nell'esempio seguente viene preparata un'applicazione che mostra la persona nella tabella principale `stageview` e crea una persona in una finestra di dialogo (`AddingPersonDialog`). GUI create da `SceneBuilder`, ma possono essere create con codici java puri.

Applicazione di esempio:

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>
                </HBox>
            </children>
        </VBox>
    </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 ();
}
}

```

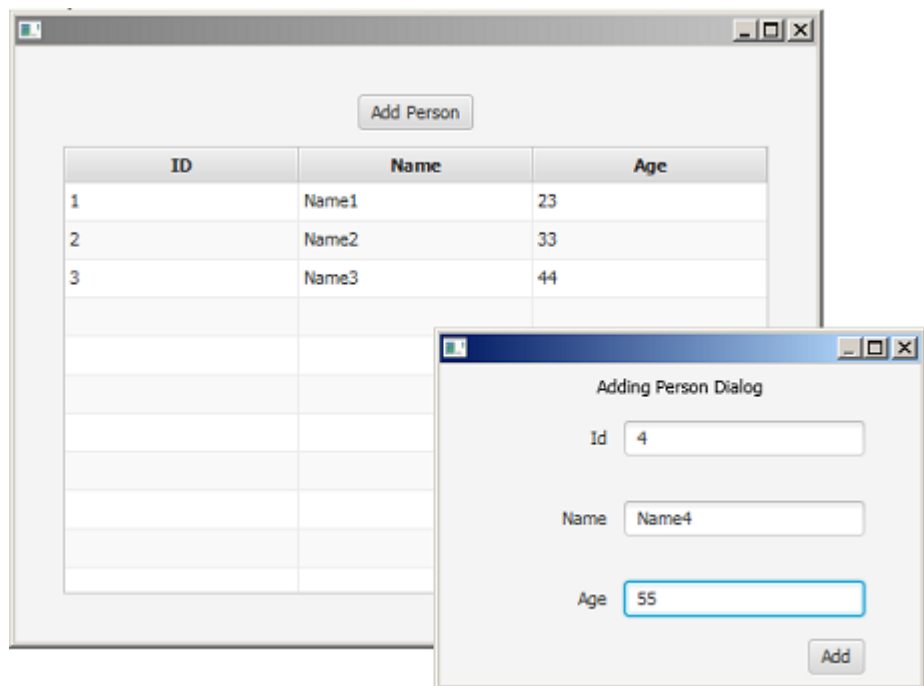


Immagine dello schermo

Creazione di finestre di dialogo personalizzate

È possibile creare finestre di dialogo personalizzate che contengono molti componenti ed eseguire molte funzionalità su di esso. Si comporta come un secondo stadio sul palco del proprietario. Nell'esempio seguente viene preparata un'applicazione che mostra la persona nella tabella principale stageview e crea una persona in una finestra di dialogo (AddingPersonDialog). GUI

create da SceneBuilder, ma possono essere create con codici java puri.

Applicazione di esempio:

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();
    }

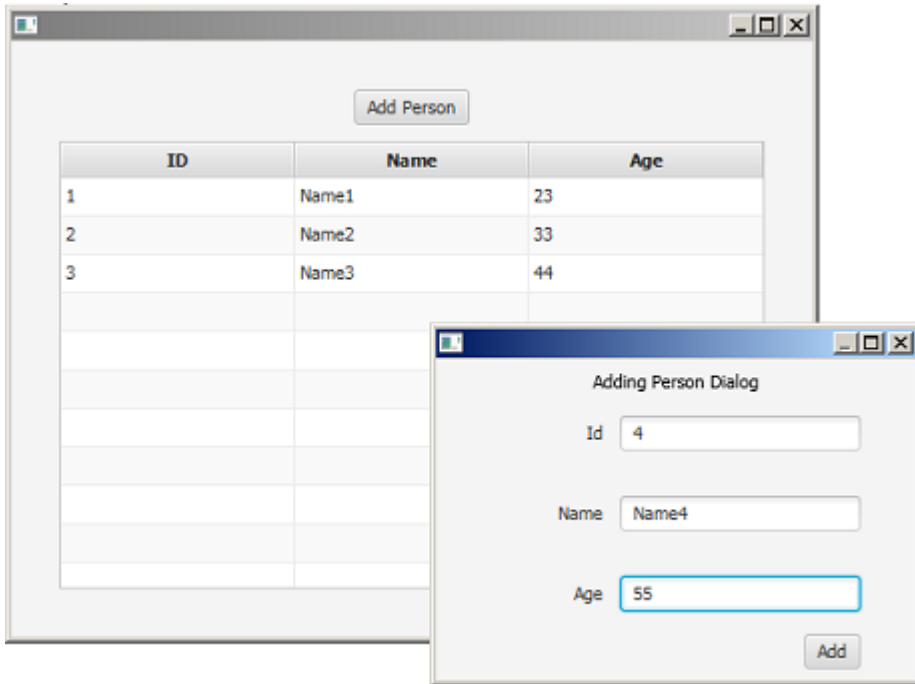
    public void setAge(int age) {
        this.age.set (age);
    }

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

}

```

Immagine dello schermo



Leggi finestre online: <https://riptutorial.com/it/javafx/topic/1496/finestre>

Capitolo 6: Finestre di dialogo

Osservazioni

Le finestre di dialogo sono state aggiunte nell'aggiornamento 40 di JavaFX 8.

Examples

TextInputDialog

`TextInputDialog` consente all'utente di chiedere all'utente di immettere una singola `String`.

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

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

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

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

ChoiceDialog

`ChoiceDialog` consente all'utente di scegliere un elemento da un elenco di opzioni.

```
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);
```

Mettere in guardia

`Alert` è un semplice popup che visualizza un insieme di pulsanti e ottiene un risultato in base al pulsante su cui l'utente ha fatto clic:

Esempio

Ciò consente all'utente di decidere, se (s) vuole davvero chiudere la fase primaria:

```
@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();
}
```

Si noti che il testo del pulsante viene regolato automaticamente in base alle `Locale`.

Testo del pulsante personalizzato

Il testo visualizzato in un pulsante può essere personalizzato, creando `ButtonType` stesso un'istanza `ButtonType`:

```
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();
```

Leggi Finestre di dialogo online: <https://riptutorial.com/it/javafx/topic/3681/finestre-di-dialogo>

Capitolo 7: FXML e controller

Sintassi

- `xmlns:fx = "http://javafx.com/fxml"` // dichiarazione dello spazio dei nomi

Examples

Esempio FXML

Un documento FXML semplice che descrive un `AnchorPane` contenente un pulsante e un nodo etichetta:

```
<?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>
```

Questo file FXML di esempio è associato a una classe controller. L'associazione tra FXML e la classe controller, in questo caso, viene effettuata specificando il nome della classe come valore dell'attributo `fx:controller` nell'elemento radice di FXML:

`fx:controller="com.example.FXMLDocumentController"` . La classe controller consente l'esecuzione del codice Java in risposta alle azioni dell'utente sugli elementi dell'interfaccia utente definiti nel file FXML:

```
package com.example ;

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

public class FXMLDocumentController {

    @FXML
    private Label label;
```

```

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

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

```

Un `FXMLLoader` può essere utilizzato per caricare il file FXML:

```

public class MyApp extends Application {

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

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

        Scene scene = new Scene(root);

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

```

Il metodo di `load` esegue diverse azioni ed è utile per capire l'ordine in cui si verificano. In questo semplice esempio:

1. `FXMLLoader` legge e analizza il file FXML. Crea oggetti corrispondenti agli elementi definiti nel file e prende nota di eventuali attributi `fx:id` definiti su di essi.
2. Poiché l'elemento radice del file FXML ha definito un attributo `fx:controller`, `FXMLLoader` crea una *nuova istanza* della classe che specifica. Per impostazione predefinita ciò avviene richiamando il costruttore no-argument sulla classe specificata.
3. Tutti gli elementi con gli attributi `fx:id` definiti che hanno campi nel controller con nomi di campi corrispondenti e che sono `public` (non consigliato) o annotati `@FXML` (consigliato) vengono "iniettati" in quei campi corrispondenti. Quindi in questo esempio, dal momento che esiste `Label` nel file FXML con `fx:id="label"` e un campo nel controller definito come

```

@FXML
private Label label ;

```

il campo `label` viene inizializzato con l'istanza `Label` creata da `FXMLLoader`.

4. I gestori di eventi sono registrati con qualsiasi elemento nel file `onXXX="#..."` con le proprietà `onXXX="#..."` definite. Questi gestori eventi invocano il metodo specificato nella classe controller. In questo esempio, poiché `Button` ha `onAction="#handleButtonAction"` e il controller definisce un metodo

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

quando un'azione viene attivata sul pulsante (ad esempio, l'utente lo preme), questo metodo viene richiamato. Il metodo deve avere un tipo restituito `void` e può definire un parametro corrispondente al tipo di evento (`ActionEvent` in questo esempio) oppure non può definire alcun parametro.

5. Infine, se la classe controller definisce un metodo di `initialize`, questo metodo viene richiamato. Si noti che questo accade dopo che i campi `@FXML` sono stati iniettati, quindi possono essere tranquillamente utilizzati con questo metodo e verranno inizializzati con le istanze corrispondenti agli elementi nel file FXML. Il metodo `initialize()` può accettare alcun parametro o può prendere un `URL` e un `ResourceBundle`. In quest'ultimo caso, questi parametri verranno popolati `URL` rappresenta la posizione del file FXML e da qualsiasi set di `ResourceBundle` su `FXMLLoader` tramite `loader.setResources(...)`. Ciascuno di questi può essere `null` se non sono stati impostati.

Controller annidati

Non è necessario creare l'intera interfaccia utente in un singolo FXML utilizzando un singolo controller.

Il `<fx:include>` può essere utilizzato per includere un file fxml in un altro. Il controller del fxml incluso può essere iniettato nel controller del file incluso come qualsiasi altro oggetto creato da `FXMLLoader`.

Questo viene fatto aggiungendo l'attributo `fx:id` all'elemento `<fx:include>`. In questo modo il controller del fxml incluso verrà iniettato nel campo con il nome `<fx:id value>Controller`.

Esempi:

| fx: valore id | nome del campo per l'iniezione |
|---------------|--------------------------------|
| foo | fooController |
| answer42 | answer42Controller |
| Xyz | xYzController |

Esempio di fxmls

contatore

Questo è un fxml che contiene uno `StackPane` con un nodo di `Text` . Il controller per questo file fxml consente di ottenere il valore del contatore corrente e di incrementare il contatore:

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;
    }
}
```

Compreso fxml

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

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

<BorderPane prefHeight="500" prefWidth="500" xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="counter.OuterController">
    <left>
        <Button BorderPane.alignment="CENTER" text="increment" onAction="#increment" />
    </left>
```

```

</left>
<center>
  <!-- content from counter.fxml included here -->
  <fx:include fx:id="count" source="counter.fxml" />
</center>
</BorderPane>

```

OuterController

Il controller del fxml incluso viene iniettato su questo controller. Qui il gestore dell'evento `onAction` per il `Button` viene utilizzato per incrementare il contatore.

```

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();
    }

}

```

Gli fxmls possono essere caricati in questo modo, assumendo che il codice sia chiamato da una classe nello stesso pacchetto di `outer.fxml` :

```

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

```

Definisci i blocchi e

A volte un elemento deve essere creato al di fuori della solita struttura di oggetti in fxml.

Qui entra in gioco *Define Blocks* :

I contenuti all'interno di un elemento `<fx:define>` non vengono aggiunti all'oggetto creato per l'elemento genitore.

Ogni elemento figlio di `<fx:define>` richiede un attributo `fx:id` .

Gli oggetti creati in questo modo possono essere successivamente referenziati usando l'elemento `<fx:reference>` o usando l'espressione binding.

L'elemento `<fx:reference>` può essere utilizzato per fare riferimento a qualsiasi elemento con un

attributo `fx:id` che viene gestito prima che l'elemento `<fx:reference>` venga gestito utilizzando lo stesso valore dell'attributo `fx:id` dell'elemento di riferimento nel attributo `source` dell'elemento `<fx:reference>` .

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

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

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

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

Trasmissione dei dati a FXML - accesso al controller esistente

Problema: alcuni dati devono essere passati a una scena caricata da un fxml.

Soluzione

Specificare un controller utilizzando l'attributo `fx:controller` e ottenere l'istanza del controller creata durante il processo di caricamento `FXMLLoader` utilizzata per caricare l'fxml.

Aggiungi metodi per passare i dati all'istanza del controller e gestisci i dati con questi metodi.

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>
```

controllore

```
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);
    }

}
```

Codice utilizzato per caricare il file fxml

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

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

Trasmissione dei dati a FXML - Specifica dell'istanza del controllore

Problema: alcuni dati devono essere passati a una scena caricata da un fxml.

Soluzione

Impostare il controller usando l'istanza `FXMLLoader` utilizzata in seguito per caricare il file fxml.

Assicurarsi che il controller contenga i dati rilevanti prima di caricare il file fxml.

Nota: in questo caso il file fxml non deve contenere l'attributo `fx:controller`.

FXML

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

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



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

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

controllore

```
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);
    }

}
```

Codice utilizzato per caricare il file fxml

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

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

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

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

Passaggio dei parametri a FXML - utilizzando un controllerFactory

Problema: alcuni dati devono essere passati a una scena caricata da un fxml.

Soluzione

Specificare una factory controller che è responsabile della creazione dei controller. Passa i dati all'istanza del controller creata dalla fabbrica.

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>

```

controllore

```

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);
    }

}

```

Codice utilizzato per caricare il file fxml

String data = "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();

```

Questo può sembrare complesso, ma può essere utile, se l'FXML dovrebbe essere in grado di decidere, quale classe di controller ha bisogno.

Creazione di istanze in FXML

La seguente classe è utilizzata per dimostrare come possono essere create istanze di classi:

JavaFX 8

L'annotazione in `Person(@NamedArg("name") String name)` deve essere rimossa, poiché l'annotazione `@NamedArg` non è disponibile.

```

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();
}
}

```

Supponiamo che la classe `Person` sia già stata inizializzata prima di caricare il file fxml.

Una nota sulle importazioni

Nel seguente esempio fxml la sezione delle importazioni sarà esclusa. Comunque il fxml dovrebbe iniziare con

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

seguito da una sezione delle importazioni che importa tutte le classi utilizzate nel file fxml. Tali importazioni sono simili alle importazioni non statiche, ma vengono aggiunte come istruzioni di elaborazione. **È necessario importare anche le classi dal pacchetto `java.lang`.**

In questo caso è necessario aggiungere le seguenti importazioni:

```

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

```

JavaFX 8

`@NamedArg` **annotato** `@NamedArg`

Se esiste un costruttore in cui ogni parametro è annotato con `@NamedArg` e tutti i valori delle annotazioni `@NamedArg` sono presenti in fxml, il costruttore verrà utilizzato con tali parametri.

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

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

```
<name>
  <String fx:value="John"/>
</name>
</Person>
```

Entrambi producono il seguente output della console, se caricato:

```
Person(String)
```

Nessun costruttore di args

Se non è disponibile un `@NamedArg` annotato `@NamedArg` adatto, `@NamedArg` utilizzato il costruttore che non accetta parametri.

Rimuovi l'annotazione `@NamedArg` dal costruttore e prova a caricare.

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

Questo userà il costruttore senza parametri.

Produzione:

```
Person()
setter
```

`fx:value` attributo `fx:value`

L'attributo `fx:value` può essere utilizzato per passare il suo valore a un metodo `valueOf` statico `valueOf` un parametro `String` e restituisce l'istanza da utilizzare.

Esempio

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

Produzione:

```
valueOf
Person(String)
```

`fx:factory`

L'attributo `fx:factory` consente la creazione di oggetti usando metodi `static` arbitrari che non prendono parametri.

Esempio

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

```
<name>
  <String fx:value="John"/>
</name>
</Person>
```

Produzione:

```
createPerson
Person()
setter
```

<fx:copy>

Usando `fx:copy` un costruttore di copia può essere invocato. Specificare `fx:id` di un altro L'attributo `source` del tag invocherà il costruttore di copia con quell'oggetto come parametro.

Esempio:

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

Produzione

```
Person(Person)
getter
```

`fx:constant`

`fx:constant` consente di ottenere un valore da un campo `static final`.

Esempio

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

non produrrà alcun output, poiché questo si riferisce solo a `JOHN` che è stato creato durante l'inizializzazione della classe.

Impostazione delle proprietà

Esistono diversi modi per aggiungere dati a un oggetto in fxml:

tag `<property>`

Un tag con il nome di una proprietà può essere aggiunto come figlio di un elemento utilizzato per creare un'istanza. Il figlio di questo tag viene assegnato alla proprietà utilizzando il setter o aggiunto al contenuto della proprietà (proprietà readonly list / map).

Proprietà di default

Una classe può essere annotata con l'annotazione `@DefaultProperty`. In questo caso gli elementi possono essere aggiunti direttamente come elemento figlio senza utilizzare un elemento con il nome della proprietà.

`property="value"` **attributo** `property="value"`

Le proprietà possono essere assegnate utilizzando il nome della proprietà come nome attributo e il valore come valore dell'attributo. Questo ha lo stesso effetto dell'aggiunta del seguente elemento come figlio del tag:

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

setter statici

Le proprietà possono essere impostate anche con setter `static`. Questi sono metodi `static` denominati `setProperty` che accettano l'elemento come primo parametro e il valore da impostare come secondo parametro. Questi metodi possono essere inseriti in qualsiasi classe e possono essere utilizzati utilizzando `ContainingClass.property` anziché il solito nome di proprietà.

Nota: attualmente sembra necessario disporre di un metodo getter statico corrispondente (ovvero un metodo statico denominato `getProperty` prende l'elemento come parametro nella stessa classe del setter statico) affinché funzioni, a meno che il tipo di valore non sia `String`.

Tipo coercizione

Il seguente meccanismo viene utilizzato per ottenere un oggetto della classe corretta durante le assegnazioni, ad esempio per soddisfare il tipo di parametro di un metodo setter.

Se le classi sono assegnabili, viene utilizzato il valore stesso.

Altrimenti il valore viene convertito come segue

| Tipo di bersaglio | valore usato (valore sorgente <i>s</i>) |
|---|---|
| <code>Boolean</code> , <code>boolean</code> | <code>Boolean.valueOf(s)</code> |
| <code>char</code> , <code>Character</code> | <code>s.toString.charAt(0)</code> |
| altro tipo | metodo appropriato per il tipo di destinazione, nel caso in cui <i>s</i> sia un |

| Tipo di bersaglio | valore usato (valore sorgente <i>s</i>) |
|-----------------------------|--|
| primitivo o tipo di wrapper | Number, il <code>valueOf(s.toString())</code> per il tipo di wrapper in caso contrario |
| BigInteger | <code>BigInteger.valueOf(s.longValue())</code> is <i>s</i> è un Number, <code>new BigInteger(s.toString())</code> altrimenti |
| BigDecimal | <code>BigDecimal.valueOf(s.doubleValue())</code> is <i>s</i> è un Number, <code>new BigDecimal(s.toString())</code> altrimenti |
| Numero | <code>Double.valueOf(s.toString())</code> se <code>s.toString()</code> contiene a . , <code>Long.valueOf(s.toString())</code> altrimenti |
| Class | <code>Class.forName(s.toString())</code> invocato utilizzando il contesto <code>ClassLoader</code> del thread corrente senza inizializzare la classe |
| enum | Il risultato del metodo <code>valueOf</code> , ulteriormente convertito in una <code>String</code> tutta maiuscola separata da <code>_</code> inserita prima di ogni lettera maiuscola, se <i>s</i> è una <code>String</code> che inizia con una lettera minuscola |
| altro | il valore restituito da un metodo <code>valueOf static</code> in <code>targetType</code> , che ha un parametro corrispondente al tipo di <i>s</i> o una superclasse di quel tipo |

Nota: questo comportamento non è ben documentato e potrebbe essere soggetto a modifiche.

Esempio

```
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;
}

}

```

Stampa il risultato del caricamento dei seguenti `FXML` file `FXML`

```

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

```

```

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

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

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

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

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

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

```

Leggi FXML e controller online: <https://riptutorial.com/it/javafx/topic/1580/fxml-e-controller>

Capitolo 8: Grafico

Examples

Grafico a torta

La classe `PieChart` disegna i dati sotto forma di cerchio diviso in sezioni. Ogni fetta rappresenta una percentuale (parte) per un valore particolare. I dati del grafico a torta sono `PieChart.Data` oggetti `PieChart.Data`. Ogni oggetto `PieChart.Data` ha due campi: il nome della fetta della torta e il suo valore corrispondente.

Costruttori

Per creare un grafico a torta, è necessario creare l'oggetto della classe `PieChart`. Due costruttori sono dati a nostra disposizione. Uno di questi crea un grafico vuoto che non mostrerà nulla a meno che i dati non siano impostati con il metodo `setData`:

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

E il secondo richiede una `ObservableList` di `PieChart.Data` da passare come parametro.

```
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
```

Dati

I valori delle sezioni di torta non devono necessariamente riassumere fino a 100, poiché la dimensione della fetta verrà calcolata in proporzione alla somma di tutti i valori.

L'ordine in cui le voci di dati vengono aggiunte all'elenco determinerà la loro posizione sul grafico. Di default sono posizionati in senso orario, ma questo comportamento può essere invertito:

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

Esempio

L'esempio seguente crea un grafico a torta semplice:

```
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);
    }
}
```

Produzione:

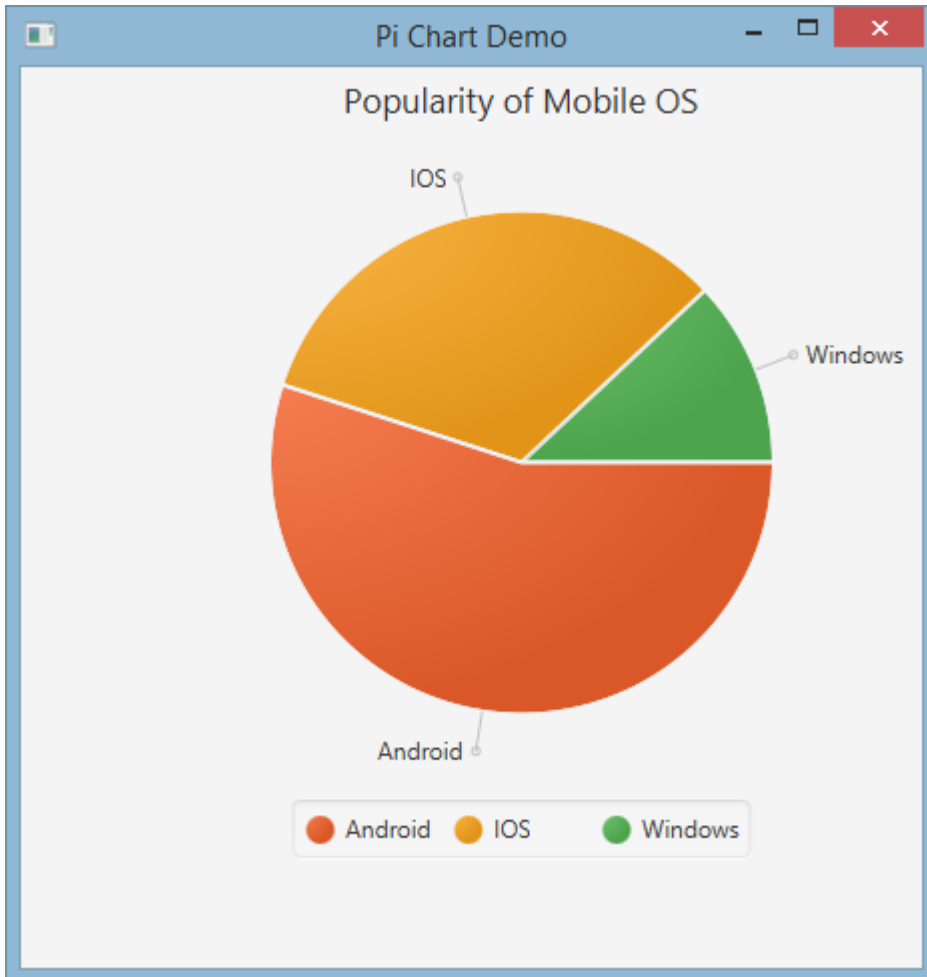


Grafico a torta interattivo

Per impostazione predefinita, `PieChart` non gestisce alcun evento, ma questo comportamento può essere modificato poiché ogni slice di torta è un `Node` JavaFX.

Nell'esempio seguente inizializziamo i dati, li assegniamo al grafico, quindi eseguiamo l'iterazione sul set di dati aggiungendo suggerimenti a ogni sezione, in modo che i valori, normalmente nascosti, possano essere presentati all'utente.

```
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);  
});
```

Grafico a linee

La classe `LineChart` presenta i dati come una serie di punti dati collegati con linee rette. Ciascun punto di dati è racchiuso nell'oggetto `XYChart.Data` e i punti di dati sono raggruppati in `XYChart.Series`.

Ogni oggetto `XYChart.Data` ha due campi, a cui è possibile accedere utilizzando `getXValue` e `getYValue`, che corrispondono a un valore x e ay su un grafico.

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

assi

Prima di creare un `LineChart` dobbiamo definire i suoi assi. Ad esempio, il costruttore predefinito senza argomenti di una classe `NumberAxis` creerà un asse auto-ranging che è pronto per l'uso e non richiede ulteriori configurazioni.

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

Esempio

Nell'esempio completo di seguito creiamo due serie di dati che verranno visualizzati sullo stesso grafico. Le etichette, gli intervalli e i valori di tick dell'asse sono esplicitamente definiti.

```
@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);

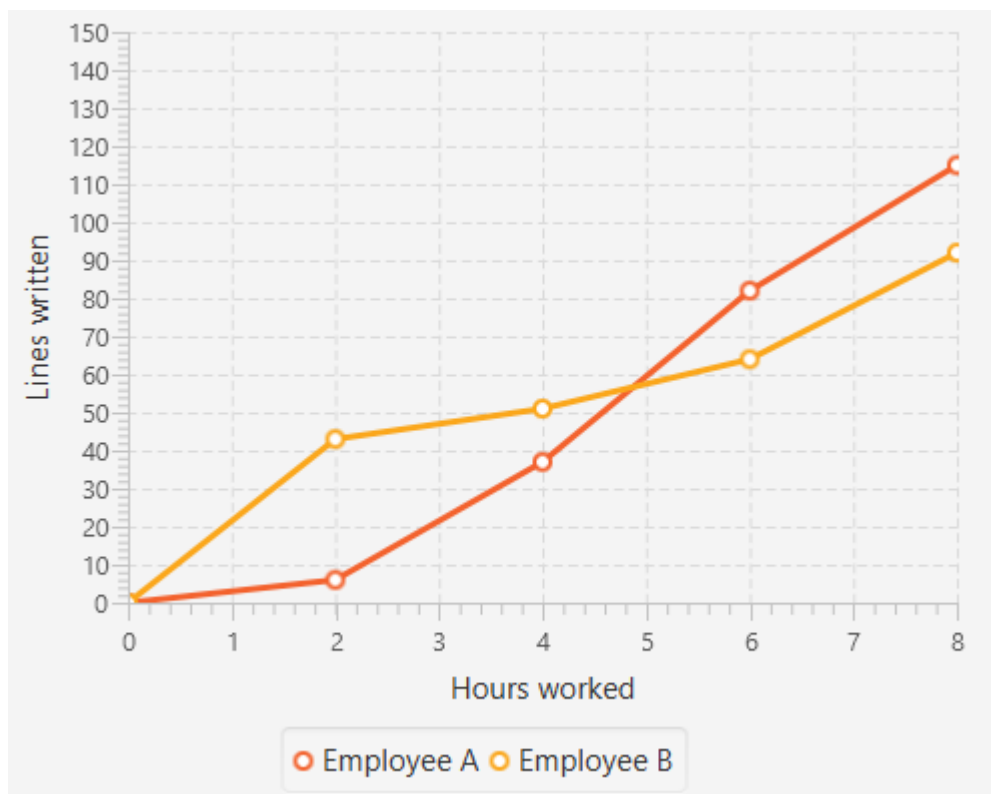
LineChart chart = new LineChart(xAxis, yAxis, seriesList);

root.getChildren().add(chart);

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

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

Produzione:



Leggi Grafico online: <https://riptutorial.com/it/javafx/topic/2631/grafico>

Capitolo 9: Internazionalizzazione in JavaFX

Examples

Caricamento del pacchetto di risorse

JavaFX offre un modo semplice per internazionalizzare le interfacce utente. Durante la creazione di una vista da un file FXML puoi fornire a `FXMLLoader` un pacchetto di risorse:

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

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

Questo pacchetto fornito viene automaticamente utilizzato per tradurre tutti i testi nel file FXML che iniziano con `%`. Diciamo che il file delle proprietà `strings_en_UK.properties` contiene la seguente riga:

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

Se hai una definizione di pulsante nel tuo FXML in questo modo:

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

Riceverà automaticamente la traduzione per la chiave `ui.button.text`.

controllore

Un pacchetto di risorse contiene oggetti specifici delle impostazioni internazionali. È possibile passare il pacchetto a `FXMLLoader` durante la sua creazione. Il controller deve implementare il metodo `Initializable` `interface` e `override` `initialize(URL location, ResourceBundle resources)`. Il secondo parametro di questo metodo è `ResourceBundle` che viene passato da `FXMLLoader` al controller e può essere utilizzato dal controller per tradurre ulteriormente i testi o modificare altre informazioni dipendenti dalla localizzazione.

```
public class MyController implements Initializable {

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

Cambio della lingua in modo dinamico quando l'applicazione è in esecuzione

Questo esempio mostra come creare un'applicazione JavaFX, in cui la lingua può essere

cambiata dinamicamente mentre l'applicazione è in esecuzione.

Questi sono i file di bundle dei messaggi utilizzati nell'esempio:

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}
```

L'idea di base è di avere una classe di utilità I18N (in alternativa potrebbe essere implementata una singleton).

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

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

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

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

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

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

```

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

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

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

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

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

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

/**
 * creates a String Binding to a localized String that is computed by calling the given
 func
 *
 * @param func
 *         function called on every change
 * @return StringBinding
 */
public static StringBinding createStringBinding(Callable<String> func) {

```

```

        return Bindings.createStringBinding(func, locale);
    }

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

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

```

Questa classe ha un'impostazione `locale` campo statica che è un oggetto `Locale` Java racchiuso in una `ObjectProperty` JavaFX, in modo da poter creare associazioni per questa proprietà. I primi metodi sono i metodi standard per ottenere e impostare una proprietà JavaFX.

Il `get(final String key, final Object... args)` è il metodo di base utilizzato per l'estrazione reale di un messaggio da un `ResourceBundle`.

I due metodi denominati `createStringBinding` creano un `StringBinding` associato al campo `locale` e quindi i binding cambieranno ogni volta che la proprietà `locale` cambia. Il primo usa i suoi argomenti per recuperare e formattare un messaggio usando il metodo `get` menzionato sopra, il secondo è passato in un `Callable`, che deve produrre il nuovo valore di stringa.

Gli ultimi due metodi sono metodi per creare componenti JavaFX. Il primo metodo viene utilizzato per creare `Label` e utilizza un `Callable` per il suo binding di stringhe interne. Il secondo crea un `Button` e utilizza un valore chiave per il recupero del collegamento stringa.

Naturalmente molti altri oggetti possono essere creati come `MenuItem` o `ToolTip` ma questi due dovrebbero essere sufficienti per un esempio.

Questo codice mostra come questa classe viene utilizzata all'interno dell'applicazione:

```

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);
    }
}

```

L'applicazione mostra tre diversi modi di utilizzare `StringBinding` creato dalla classe `I18N` :

1. il titolo della finestra è associato direttamente utilizzando `StringBinding` .
2. i pulsanti usano il metodo di supporto con i tasti dei messaggi
3. l'etichetta usa il metodo di supporto con un `Callable` . Questo `Callable` utilizza il metodo `I18N.get ()` per ottenere una stringa tradotta formattata contenente il conteggio effettivo delle opzioni.

Facendo clic su un pulsante, il contatore viene aumentato e viene impostata la proprietà locale `I18N` , che a sua volta attiva la modifica delle stringhe e quindi imposta la stringa dell'interfaccia utente su nuovi valori.

Leggi [Internazionalizzazione in JavaFX online](https://riptutorial.com/it/javafx/topic/5434/internazionalizzazione-in-javafx):

<https://riptutorial.com/it/javafx/topic/5434/internazionalizzazione-in-javafx>

Capitolo 10: layout

Examples

StackPane

`StackPane` espone i suoi figli in uno stack back-to-front.

L'ordine z dei bambini è definito dall'ordine dell'elenco dei bambini (accessibile chiamando `getChildren`): il 0° bambino è l'ultimo e l'ultimo bambino in cima allo stack.

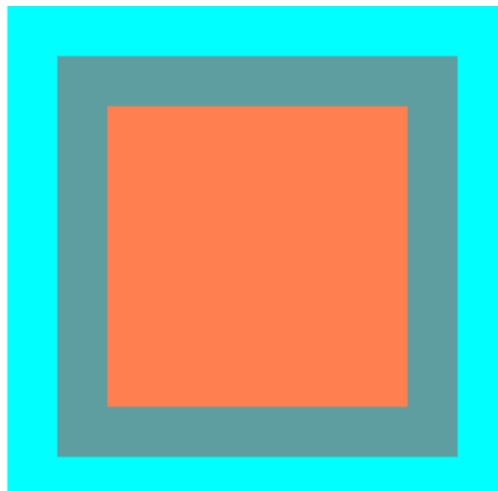
Lo stackpane tenta di ridimensionare ogni bambino per riempire la propria area di contenuto. Nel caso in cui un bambino non possa essere ridimensionato per riempire l'area dello `StackPane` (o perché non era ridimensionabile o la sua dimensione massima lo ha impedito) allora sarà allineato all'interno dell'area usando la proprietà `alignmentProperty` dello stackpane, che di default è `Pos.CENTER`.

Esempio

```
// 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 e VBox

I layout di `HBox` e `VBox` sono molto simili, entrambi presentano i loro figli in una singola riga.

Caratteristiche comuni

Se un `HBox` o un `VBox` hanno un bordo e / o un padding set, i contenuti saranno disposti all'interno di questi riquadri.

Distribuiscono ogni bambino gestito a prescindere dal valore della proprietà visibile del bambino; i bambini non gestiti vengono ignorati.

L'allineamento del contenuto è controllato dalla proprietà di allineamento, che per impostazione predefinita è `Pos.TOP_LEFT`.

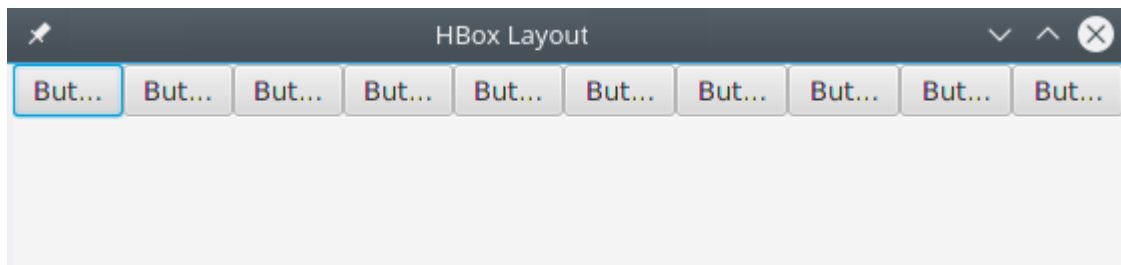
HBox

`HBox` espone i suoi figli in un'unica fila orizzontale da sinistra a destra.

`HBox` ridimensiona i bambini (se ridimensionabili) **alla loro larghezza preferita** e usa la proprietà `fillHeight` per determinare se ridimensionare le loro altezze per riempire la propria altezza o mantenere le loro altezze a loro preferite (`fillHeight` defaults su `true`).

Creare un HBox

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



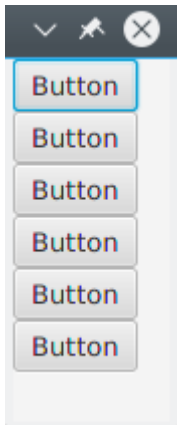
VBox

`VBox` espone i suoi figli in una singola colonna verticale dall'alto verso il basso.

`VBox` ridimensiona i bambini (se ridimensionabili) **alle loro altezze preferite** e utilizza la proprietà `fillWidth` per determinare se ridimensionare le loro larghezze per riempire la propria larghezza o mantenere le loro larghezze al loro preferito (`fillWidth` defaults su `true`).

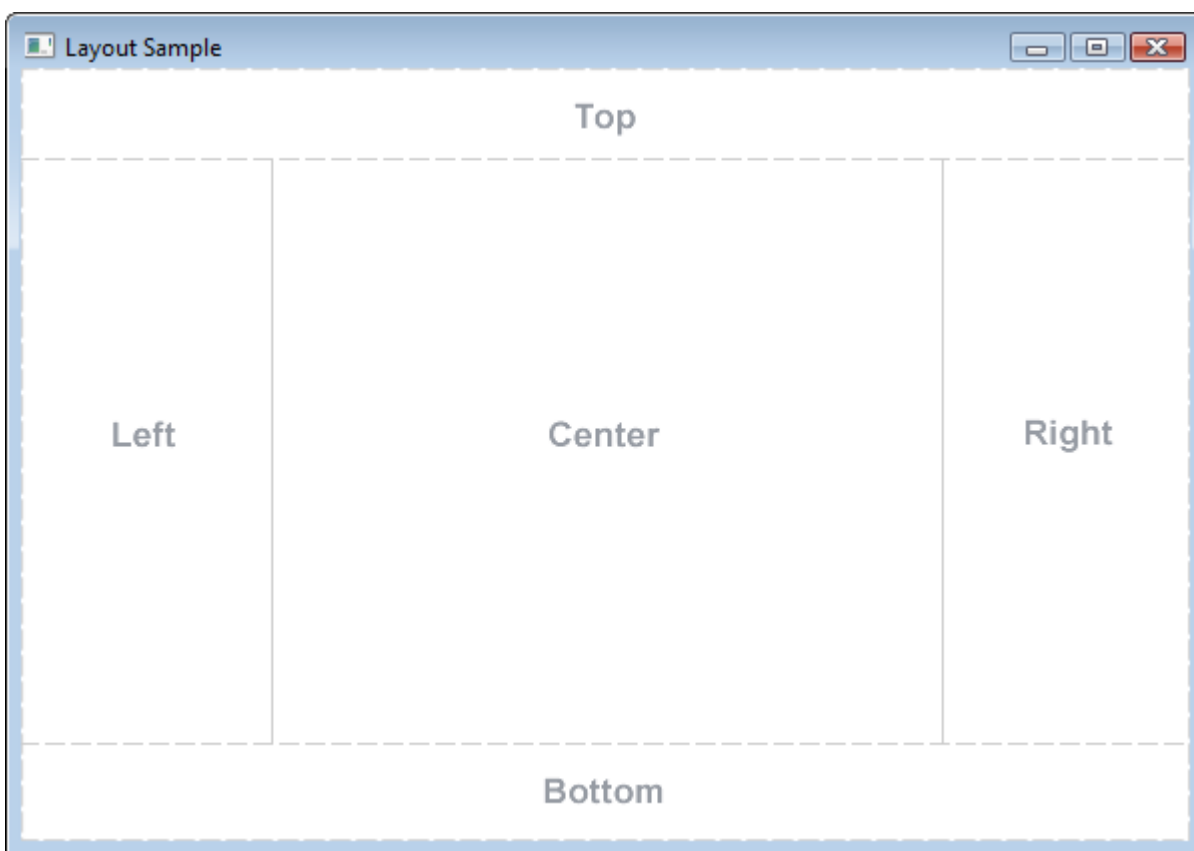
Creare un VBox

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



BorderPane

`BorderPane` è suddiviso in cinque aree diverse.



Le aree di confine (in `Top` , a `Right` , in `Bottom` , a `Left`) hanno dimensioni preferite in base al loro contenuto. Di default prenderanno solo ciò di cui hanno bisogno, mentre l'area del `Center` occuperà tutto lo spazio rimanente. Quando le aree di confine sono vuote, non occupano spazio.

Ogni area può contenere solo un elemento. Può essere aggiunto usando i metodi `setTop(Node)` , `setRight(Node)` , `setBottom(Node)` , `setLeft(Node)` , `setCenter(Node)` . Puoi usare altri layout per inserire più di un elemento in una singola area.

```
//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

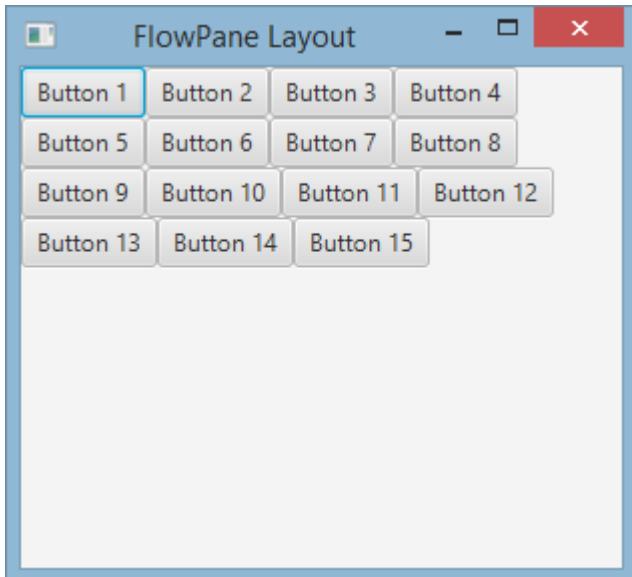
FlowPane definisce i nodi in righe o colonne in base allo spazio orizzontale o verticale disponibile. Racchiude i nodi alla riga successiva quando lo spazio orizzontale è inferiore al totale di tutte le larghezze dei nodi; esso avvolge i nodi alla colonna successiva quando lo spazio verticale è inferiore al totale di tutte le altezze dei nodi. Questo esempio illustra il layout orizzontale predefinito:

```

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);
    }
}

```



Costruttore di default `FlowPane` :

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

FlowPane aggiuntivi di `FlowPane` :

```
FlowPane() //Creates a horizontal FlowPane layout with hgap/vgap = 0 by default.
FlowPane(double hgap, double vgap) //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 vgap) //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.
```

L'aggiunta di nodi al layout utilizza i metodi `add()` **o** `addAll()` **del** `Pane` **padre:**

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

Per impostazione predefinita, un `FlowPane` espone i nodi figlio da sinistra a destra. Per modificare l'allineamento del flusso, chiamare il metodo `setAlignment()` passando un valore enumerato di tipo

`Pos` .

Alcuni allineamenti di flusso comunemente usati:

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

GridPane

`GridPane` espone i suoi figli all'interno di una griglia flessibile di righe e colonne.

Figli della GridPane

Un bambino può essere posizionato ovunque all'interno di `GridPane` e può estendersi su più righe / colonne (lo span predefinito è 1) e il suo posizionamento all'interno della griglia è definito dai vincoli di layout:

| costrizione | Descrizione |
|--------------------------|---|
| <code>columnIndex</code> | colonna in cui inizia l'area di layout del bambino. |
| <code>rowIndex</code> | riga in cui inizia l'area di layout del bambino. |
| <code>columnSpan</code> | il numero di colonne che l'area di layout del bambino si estende orizzontalmente. |
| <code>rowSpan</code> | il numero di righe dell'area di layout del bambino si estende in verticale. |

Non è necessario specificare il numero totale di righe / colonne in primo piano in quanto la griglia espanderà / contrarrà automaticamente la griglia per adattarla al contenuto.

Aggiunta di bambini a GridPane

Per aggiungere nuovi `Node` a un `GridPane` i **vincoli di layout** sui bambini devono essere impostati utilizzando il metodo statico della classe `GridPane`, quindi quei bambini possono essere aggiunti a un'istanza `GridPane`.

```
GridPane gridPane = new GridPane();

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

`GridPane` fornisce metodi convenienti per combinare questi passaggi:

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

La classe `GridPane` fornisce anche metodi di settaggio statico per impostare il **numero di righe e colonne** di elementi figli:

```
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
```

Dimensione di colonne e righe

Per impostazione predefinita, righe e colonne verranno ridimensionate per adattarsi al loro contenuto. In caso di necessità del **controllo esplicito delle dimensioni di righe e colonne**, è possibile aggiungere `RowConstraints` e `ColumnConstraints` istanze a `GridPane`. L'aggiunta di questi due vincoli ridimensiona l'esempio sopra per avere la prima colonna 100 pixel, la seconda colonna lunga 200 pixel.

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

Di default `GridPane` ridimensiona righe / colonne alle loro dimensioni preferite anche se il reticolo viene ridimensionato più grande della sua dimensione preferita. Per supportare dimensioni **dinamiche di colonne / righe**, entrambe le classi di constraints forniscono tre proprietà: dimensione minima, dimensione massima e dimensione preferita.

Inoltre `ColumnConstraints` fornisce `setHGrow` e `RowConstraints` fornisce i metodi `setVGrow` per **influenzare la priorità della crescita e della riduzione**. Le tre priorità predefinite sono:

- **Priorità . SEMPRE**: cerca sempre di aumentare (o ridurre), condividendo l'aumento (o la diminuzione) nello spazio con altre aree di layout che hanno una crescita (o restringimento) di SEMPRE
- **Priorità.SOMETIMES**: se non ci sono altre aree di layout con grow (o shrink) impostato su ALWAYS o quelle aree di layout non hanno assorbito tutto lo spazio aumentato (o diminuito), quindi condivideranno l'aumento (o la diminuzione) nello spazio con altre aree di layout di SOMETIMES.
- **Priorità.NEVER**: l'area di layout non crescerà mai (o si restringerà) quando si verifica un aumento (o una diminuzione) nello spazio disponibile nella regione.

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

La colonna sopra definita ha una dimensione minima di 100 pixel e cercherà sempre di crescere fino a raggiungere la larghezza massima di 300 pixel.

È anche possibile definire il **dimensionamento percentuale** per righe e colonne. L'esempio seguente definisce una `GridPane` cui la prima colonna riempie il 40% della larghezza della griglia, la seconda riempie il 60%.

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

Allineamento di elementi all'interno delle celle della griglia

L'allineamento di `Node` s può essere definita utilizzando il `setHalignment` metodo (orizzontale) di `ColumnConstraints` classe e `setValignment` metodo (verticale) del `RowConstraints` classe.

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

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

TilePane

Il layout del riquadro di tile è simile al layout di FlowPane. TilePane colloca tutti i nodi in una griglia in cui ogni cella o tessera ha le stesse dimensioni. Organizza nodi in righe e colonne ordinate, sia orizzontalmente che verticalmente.

```
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 {

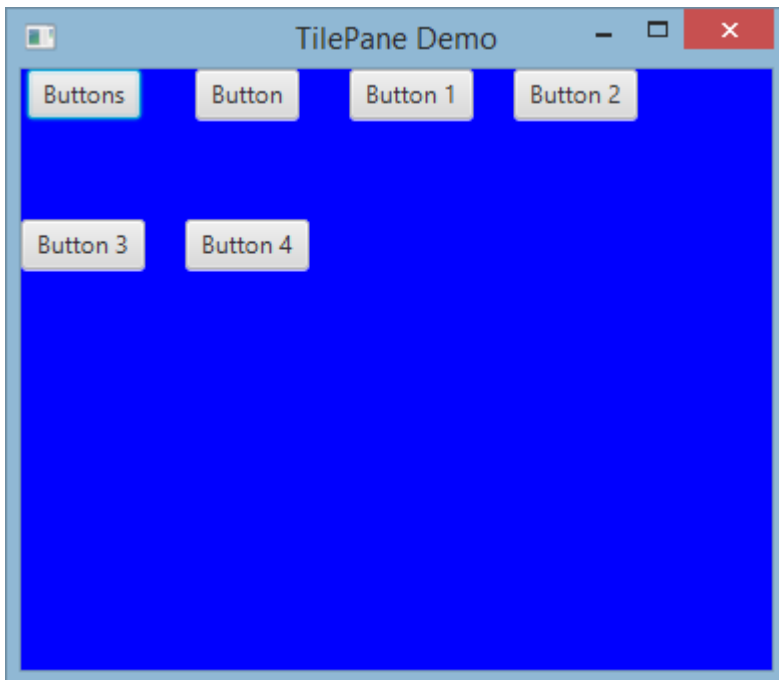
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("TilePane Demo");
        double width = 400;
        double height = 300;
        TilePane root = new TilePane();
        root.setStyle("-fx-background-color:blue");
        // to set horizontal and vertical gap
        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);
    }
}
```

```
}  
}
```

produzione



Per creare Tilepane

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

setHgap () E il metodo setVgap () viene utilizzato per creare spazi tra colonna e colonna. possiamo anche impostare le colonne per il layout usando

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

AnchorPane

`AnchorPane` a è un layout che consente di posizionare il contenuto a una distanza specifica dai lati.

Esistono 4 metodi per l'impostazione e 4 metodi per ottenere le distanze in `AnchorPane` . Il primo parametro di questi metodi è il `Node` figlio. Il secondo parametro dei setter è il valore `Double` da utilizzare. Questo valore può essere `null` senza indicare alcun vincolo per il lato specificato.

| metodo setter | metodo getter |
|-----------------|-----------------|
| setBottomAnchor | getBottomAnchor |
| setLeftAnchor | getLeftAnchor |
| setRightAnchor | getRightAnchor |

| metodo setter | metodo getter |
|---------------|---------------|
| setTopAnchor | getTopAnchor |

Nell'esempio seguente posiziona i nodi a distanze specificate dai lati.

Anche la regione `center` viene ridimensionata per mantenere le distanze specificate dai lati. Osservare il comportamento quando la finestra viene ridimensionata.

```
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();
}
```

Leggi layout online: <https://riptutorial.com/it/javafx/topic/2121/layout>

Capitolo 11: paginatura

Examples

Creazione di una paginazione

Le impaginazioni in JavaFX utilizzano un callback per ottenere le pagine utilizzate nell'animazione.

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

Questo crea una lista infinita di pulsanti `numbered 0..` poiché il costruttore zero arg crea un'impaginazione infinita. `setPageFactory` accetta un callback che accetta un `int` e restituisce il nodo che vogliamo a quell'indice.

Avanzamento automatico

```
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();
```

Questo fa avanzare l'impaginazione ogni 5 secondi.

Come funziona

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

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

`fiveSecondsWonder` è una sequenza temporale che `fiveSecondsWonder` un evento ogni volta che termina un ciclo. In questo caso il tempo di ciclo è di 5 secondi.

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

Spunta l'impaginazione.

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


Imposta la sequenza temporale per l'esecuzione per sempre.

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

Crea una paginazione di immagini

```
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)));
```

Si noti che i percorsi devono essere URL, non percorsi del file system.

Come funziona

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

Tutto il resto è semplicemente fluff, è qui che sta accadendo il vero lavoro. `setPageFactory` accetta un callback che accetta un int e restituisce il nodo che vogliamo a quell'indice. La prima pagina viene mappata al primo elemento nell'elenco, il secondo al secondo elemento nell'elenco e così via.

Leggi paginatura online: <https://riptutorial.com/it/javafx/topic/5165/paginatura>

Capitolo 12: Proprietà e osservabili

Osservazioni

Le proprietà sono osservabili e gli ascoltatori possono essere aggiunti a loro. Sono costantemente utilizzati per le proprietà dei `Node`.

Examples

Tipi di proprietà e denominazione

Proprietà standard

A seconda del tipo di proprietà, ci sono fino a 3 metodi per una singola proprietà. Lascia che `<property>` denoti il nome di una proprietà e `<Property>` il nome della proprietà con una prima lettera maiuscola. E sia `T` il tipo di proprietà; per i wrapper primitivi usiamo il tipo primitivo qui, ad es. `String` per `StringProperty` e `double` per `ReadOnlyDoubleProperty`.

| Nome del metodo | parametri | Tipo di reso | Scopo |
|---------------------------------------|------------------|---|---|
| <code><property>Property</code> | <code>()</code> | La proprietà stessa, ad es <code>DoubleProperty</code> , <code>ReadOnlyStringProperty</code> , <code>ObjectProperty<VPos></code> | restituire la proprietà stessa per l'aggiunta di listener / binding |
| <code>get<Property></code> | <code>()</code> | <code>T</code> | restituire il valore racchiuso nella proprietà |
| <code>set<Property></code> | <code>(T)</code> | <code>void</code> | imposta il valore della proprietà |

Notare che il setter non esiste per proprietà readonly.

Visualizza le proprietà in sola lettura

Le proprietà elenco di `ReadOnly` sono proprietà che forniscono solo un metodo `getter`. Il tipo di tale proprietà è `ObservableList`, preferibilmente con un tipo `argument` specificato. Il valore di questa proprietà non cambia mai; il contenuto di `ObservableList` può essere modificato.

Visualizza di sola lettura le proprietà

Analogamente all'elenco di proprietà in sola lettura, le proprietà della mappa `readonly` forniscono solo un `getter` e il contenuto può essere modificato al posto del valore della proprietà. Il `getter`

restituisce una `ObservableMap` .

Esempio `StringProperty`

L'esempio seguente mostra la dichiarazione di una proprietà (`StringProperty` in questo caso) e illustra come aggiungere un `ChangeListener` ad esso.

```
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");
    }
}
```

Esempio `ReadOnlyIntegerProperty`

Questo esempio mostra come utilizzare una proprietà wrapper readonly per creare una proprietà a cui non è possibile scrivere. In questo caso, il `cost` e il `price` possono essere modificati, ma il `profit` sarà sempre il `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);  
}  
  
}
```

Leggi Proprietà e osservabili online: <https://riptutorial.com/it/javafx/topic/4436/proprieta-e-osservabili>

Capitolo 13: Pulsante

Examples

Aggiunta di un listener di azioni

I pulsanti attivano gli eventi di azione quando vengono attivati (ad es. Si fa clic, una combinazione di tasti per il pulsante viene premuto, ...).

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

Se si utilizza Java 8+, è possibile utilizzare lambdas per i listener di azioni.

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

Aggiunta di un elemento grafico a un pulsante

I pulsanti possono avere un grafico. `graphic` può essere qualsiasi nodo JavaFX, come un `ProgressBar`

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

Un `ImageView`

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

O anche un altro pulsante

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

Crea un pulsante

La creazione di un `Button` è semplice:

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

Questo creerà un nuovo `Button` senza testo o grafica all'interno.

Se vuoi creare un `Button` con un testo, usa semplicemente il costruttore che accetta un parametro

`String` come parametro (che imposta la proprietà `textProperty` del `Button`):

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

Se si desidera creare un `Button` con un elemento grafico all'interno o qualsiasi altro `Node`, utilizzare questo costruttore:

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

Pulsanti predefiniti e Annulla

`Button` API `Button` offre un modo semplice per assegnare scorciatoie da tastiera comuni ai pulsanti senza la necessità di accedere all'elenco degli acceleratori assegnato a `Scene` o di ascoltare esplicitamente gli eventi dei tasti. Vale a dire, sono disponibili due metodi di convenienza:

`setDefaultButton` e `setCancelButton`:

- L'impostazione di `setDefaultButton` su `true` farà sì che il `Button` si `KeyCode.ENTER` ogni volta che riceve un evento `KeyCode.ENTER`.
- Se si `setCancelButton` su `true`, il `Button` viene `KeyCode.ESCAPE` ogni volta che riceve un evento `KeyCode.ESCAPE`.

L'esempio seguente crea una `Scene` con due pulsanti che vengono attivati quando vengono premuti i tasti di invio o di escape, indipendentemente dal fatto che siano focalizzati o meno.

```
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);
```

Il codice sopra non funzionerà se questi `KeyEvents` sono consumati da qualsiasi `Node` genitore:

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

Leggi Pulsante online: <https://riptutorial.com/it/javafx/topic/5162/pulsante>

Capitolo 14: Pulsante radio

Examples

Creazione di pulsanti radio

I pulsanti di scelta consentono all'utente di scegliere un elemento tra quelli indicati. Esistono due modi per dichiarare un `RadioButton` con un testo oltre a esso. O utilizzando il costruttore predefinito `RadioButton()` e impostando il testo con il `setText(String)` o utilizzando l'altro costruttore `RadioButton(String)`.

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

Poiché `RadioButton` è un'estensione di `Labeled`, può anche esserci `Image` specificata per `RadioButton`. Dopo aver creato il `RadioButton` con uno dei costruttori, aggiungi semplicemente l'`Image` con il `setGraphic(ImageView)` come qui:

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

Utilizza i gruppi sui pulsanti di opzione

Un gruppo `ToggleGroup` viene utilizzato per gestire i `RadioButton` modo che sia possibile selezionare solo uno in ciascun gruppo in ogni momento.

Crea un semplice `ToggleGroup` come segue:

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

Dopo aver creato un `ToggleGroup`, può essere assegnato ai `RadioButton` usando `setToggleGroup(ToggleGroup)`. Usa `setSelected(Boolean)` per preselezionare uno dei `RadioButton`.

```
RadioButton radioButton1 = new RadioButton("stackoverflow 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);
```

Eventi per i pulsanti radio

In genere, quando viene selezionato uno dei `RadioButton` in un gruppo `ToggleGroup` l'applicazione esegue un'azione. Di seguito è riportato un esempio che stampa i dati utente del `RadioButton` selezionato che è stato impostato con `setUserData(Object)` .

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

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

Richiesta di messa a fuoco per pulsanti di opzione

Diciamo che il secondo `RadioButton` su tre è pre-selezionato con `setSelected(Boolean)` , lo stato `setSelected(Boolean)` è ancora al primo `RadioButton` per impostazione predefinita. Per cambiarlo usa il metodo `requestFocus()` .

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

Leggi Pulsante radio online: <https://riptutorial.com/it/javafx/topic/5906/pulsante-radio>

Capitolo 15: Scene Builder

introduzione

JavaFX Scene Builder è uno strumento di layout visivo che consente agli utenti di progettare rapidamente interfacce utente di applicazioni JavaFX, senza codifica. È usato per generare file FXML.

Osservazioni

JavaFX Scene Builder è uno strumento di layout visivo che consente agli utenti di progettare rapidamente interfacce utente di applicazioni JavaFX, senza codifica. Gli utenti possono trascinare i componenti dell'interfaccia utente su un'area di lavoro, modificarne le proprietà, applicare i fogli di stile e il codice FXML per il layout che stanno creando viene generato automaticamente in background. Il risultato è un file FXML che può quindi essere combinato con un progetto Java vincolando l'interfaccia utente alla logica dell'applicazione.

Da una prospettiva Model View Controller (MVC):

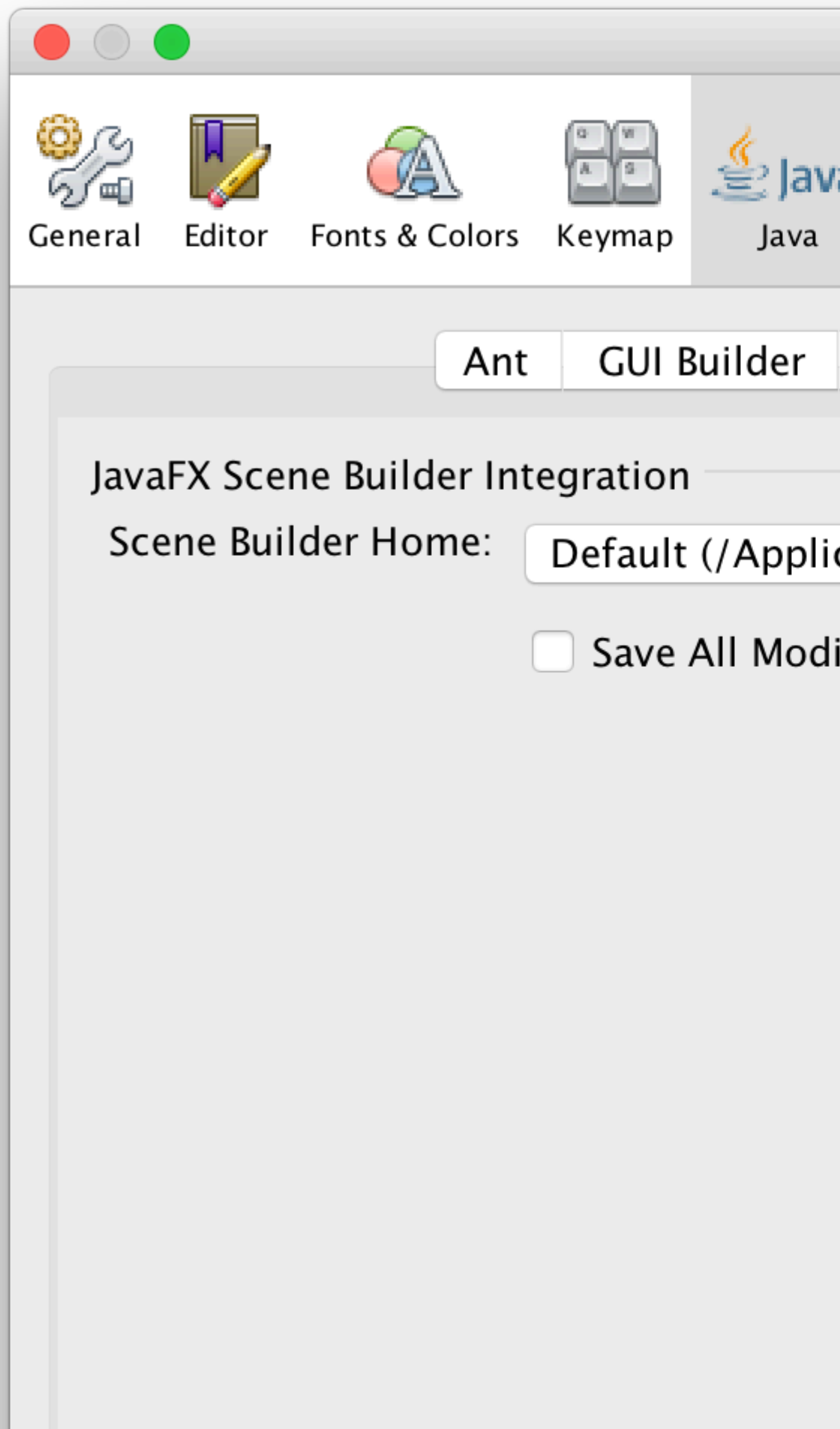
- Il file FXML, contenente la descrizione dell'interfaccia utente, è la vista.
- Il controller è una classe Java, che implementa facoltativamente la classe `Initializable`, che viene dichiarata come controller per il file FXML.
- Il modello è costituito da oggetti di dominio, definiti sul lato Java, che possono essere collegati alla vista attraverso il controller.

Installazione di Scene Builder

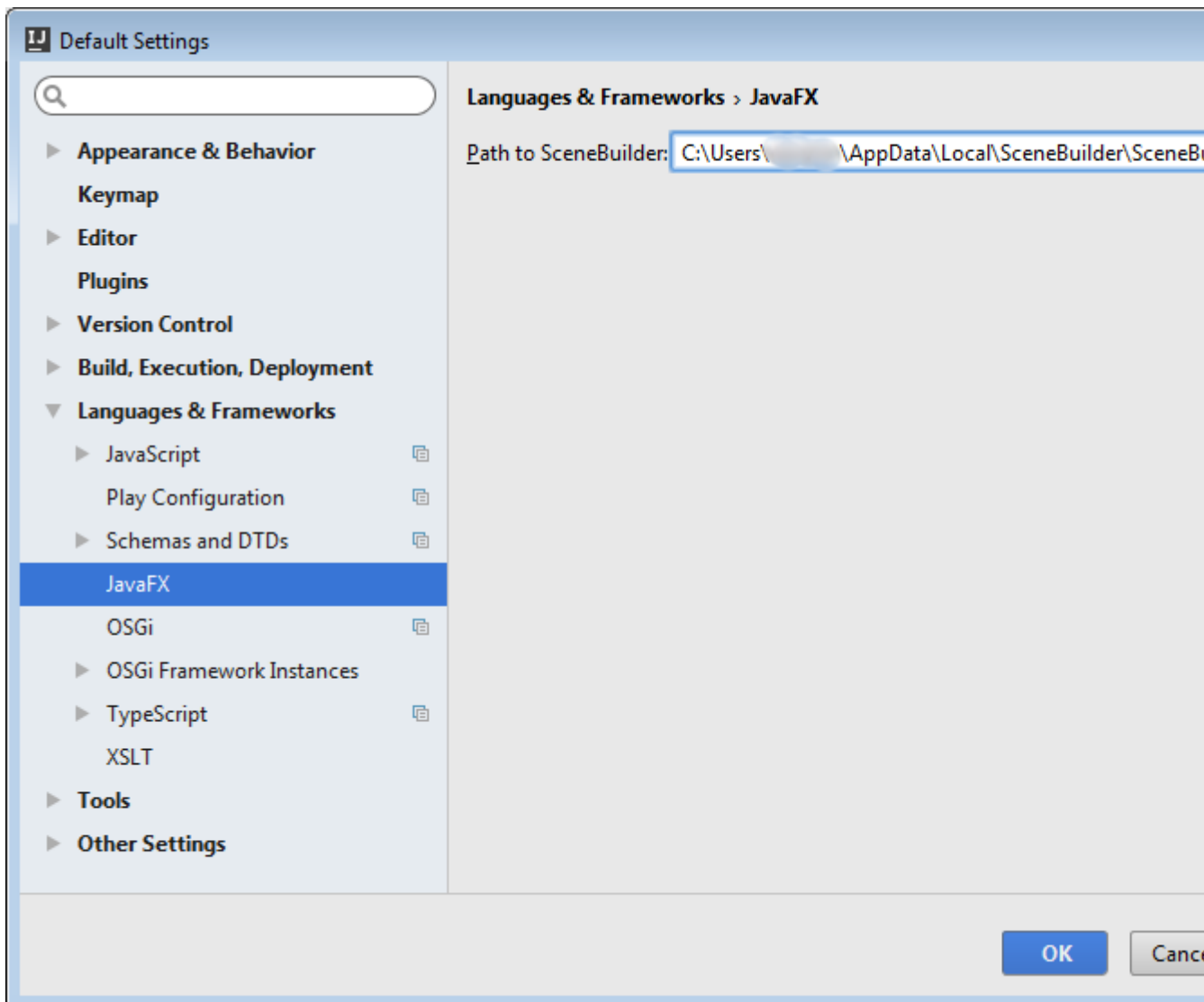
1. Scarica la versione più recente di Scene Builder dal [sito Web](#) di Gluon, selezionando l'installer per la tua piattaforma o il jar eseguibile.
2. Con il programma di installazione scaricato, fai doppio clic per installare Scene Builder sul tuo sistema. È incluso un JRE aggiornato.
3. Fare doppio clic sull'icona di Scene Builder per eseguirlo come applicazione autonoma.
4. Integrazione IDE

Mentre Scene Builder è un'applicazione standalone, produce file FXML che sono integrati con un progetto Java SE. Quando si crea questo progetto su un IDE, è utile includere un collegamento al percorso di Scene Builder, in modo che i file FXML possano essere modificati.

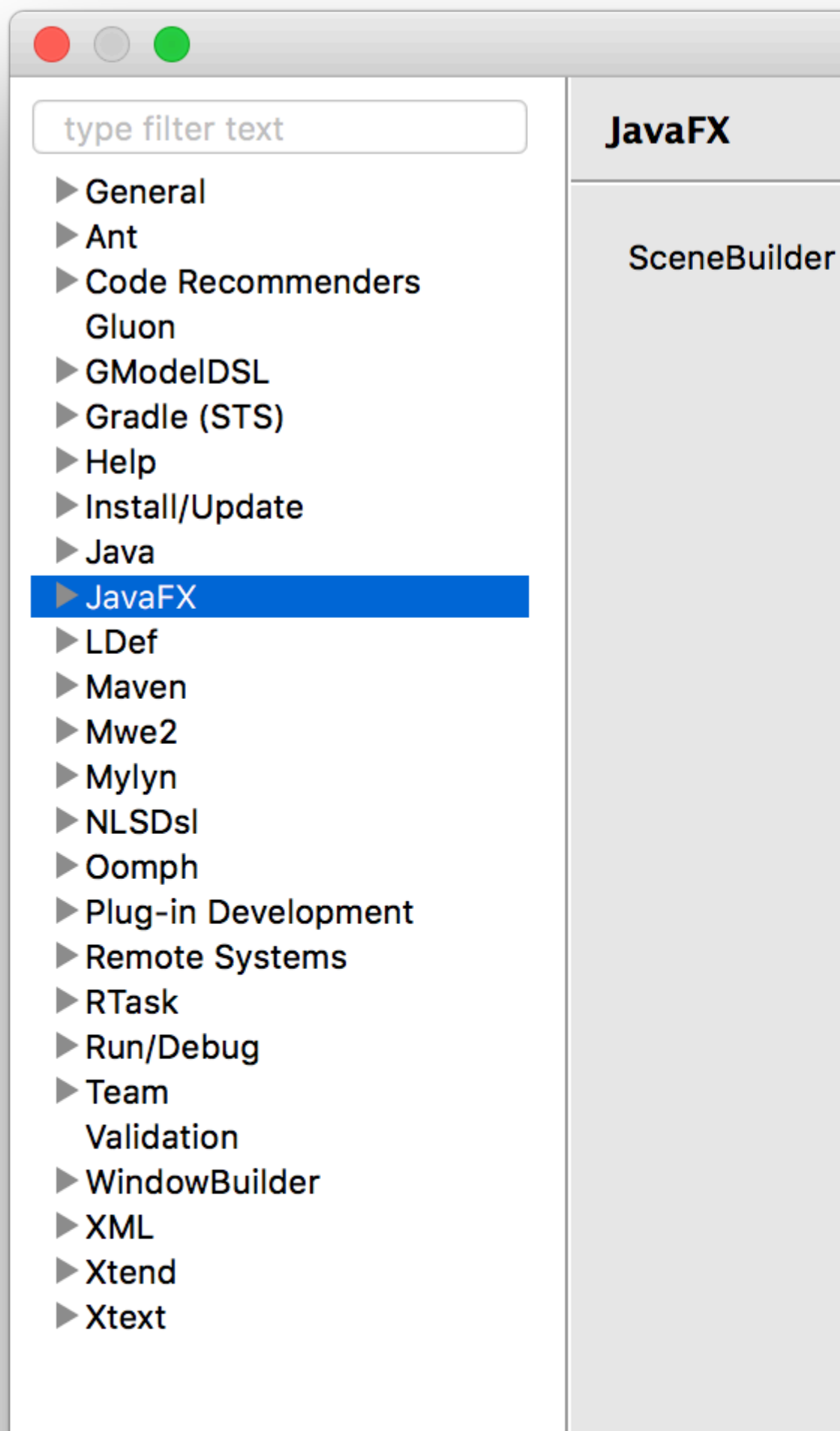
- NetBeans: in Windows vai a NetBeans-> Strumenti-> Opzioni-> Java-> JavaFX. Su Mac OS X vai su NetBeans-> Preferenze-> Java-> JavaFX. Fornire il percorso per la Home page di Scene Builder.



- IntelliJ: Su Windows vai su IntelliJ-> Impostazioni-> Lingue e quadri-> JavaFX. Su Mac OS X vai su IntelliJ-> Preferenze-> Lingue e quadri-> JavaFX. Fornire il percorso per la Home page di Scene Builder.



- Eclipse: su Windows vai su Eclipse-> Finestra-> Preferenze-> JavaFX. Su Mac OS X vai su Eclipse-> Preferenze-> JavaFX. Fornire il percorso per la Home page di Scene Builder.



binari, fino a Scene Builder v 2.0, incluse solo le funzionalità JavaFX prima del rilascio di Java SE 8u40, quindi le nuove funzionalità come i controlli `Spinner` non sono incluse.

Gluon ha [acquisito](#) la distribuzione di versioni binarie e da [qui](#) è possibile scaricare uno Scene Builder 8+ aggiornato per ogni piattaforma.

Include le ultime modifiche in JavaFX e anche miglioramenti recenti e correzioni di errori.

Il progetto open source può essere trovato [qui](#) dove è possibile creare problemi, richieste di funzionalità e richieste di pull.

I binari legacy Oracle possono ancora essere scaricati da [qui](#).

Esercitazioni

I tutorial di Scene Builder sono disponibili qui:

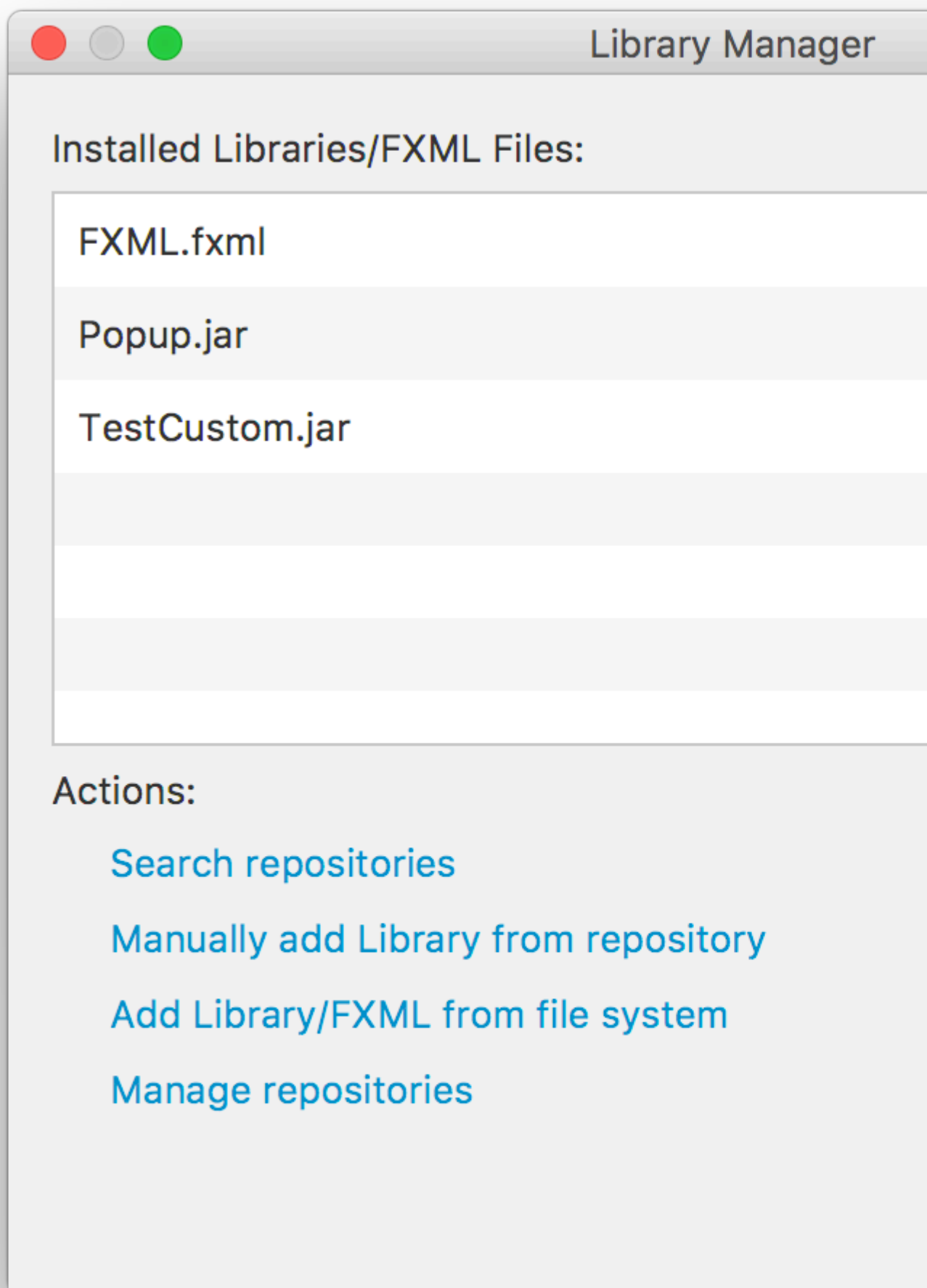
- [Tutorial di Oracle Scene Builder 2.0](#)

Le esercitazioni FXML possono essere trovate qui.

- Oracle FXML [esercitazione](#)

Controlli personalizzati

Gluon ha pienamente [documentato](#) la nuova funzionalità che consente di importare giare di terze parti con controlli personalizzati, utilizzando il Gestore libreria (disponibile da Scene Builder 8.2.0).



dal jar / classpath, come specificato da `FXMLLoader.load(getClass().getResource("BasicFXML.fxml"))`

Quando si carica `basicFXML.fxml`, il loader troverà il nome della classe controller, come specificato da `fx:controller="org.stackoverflow.BasicFXMLController"` in FXML.

Quindi il loader creerà un'istanza di quella classe, in cui proverà a iniettare tutti gli oggetti che hanno un `fx:id` nel FXML e sono contrassegnati con l'annotazione `@FXML` nella classe controller.

In questo esempio, `FXMLLoader` creerà l'etichetta basata su `<Label ... fx:id="label"/>` e inietterà l'istanza `@FXML private Label label;`.

Infine, quando l'intero FXML è stato caricato, `FXMLLoader` chiamerà il metodo di `initialize` del controller e verrà eseguito il codice che registra un gestore di azioni con il pulsante.

La modifica

Sebbene il file FXML possa essere modificato all'interno dell'IDE, non è consigliabile, poiché l'IDE fornisce solo il controllo sintattico di base e il completamento automatico, ma non la guida visiva.

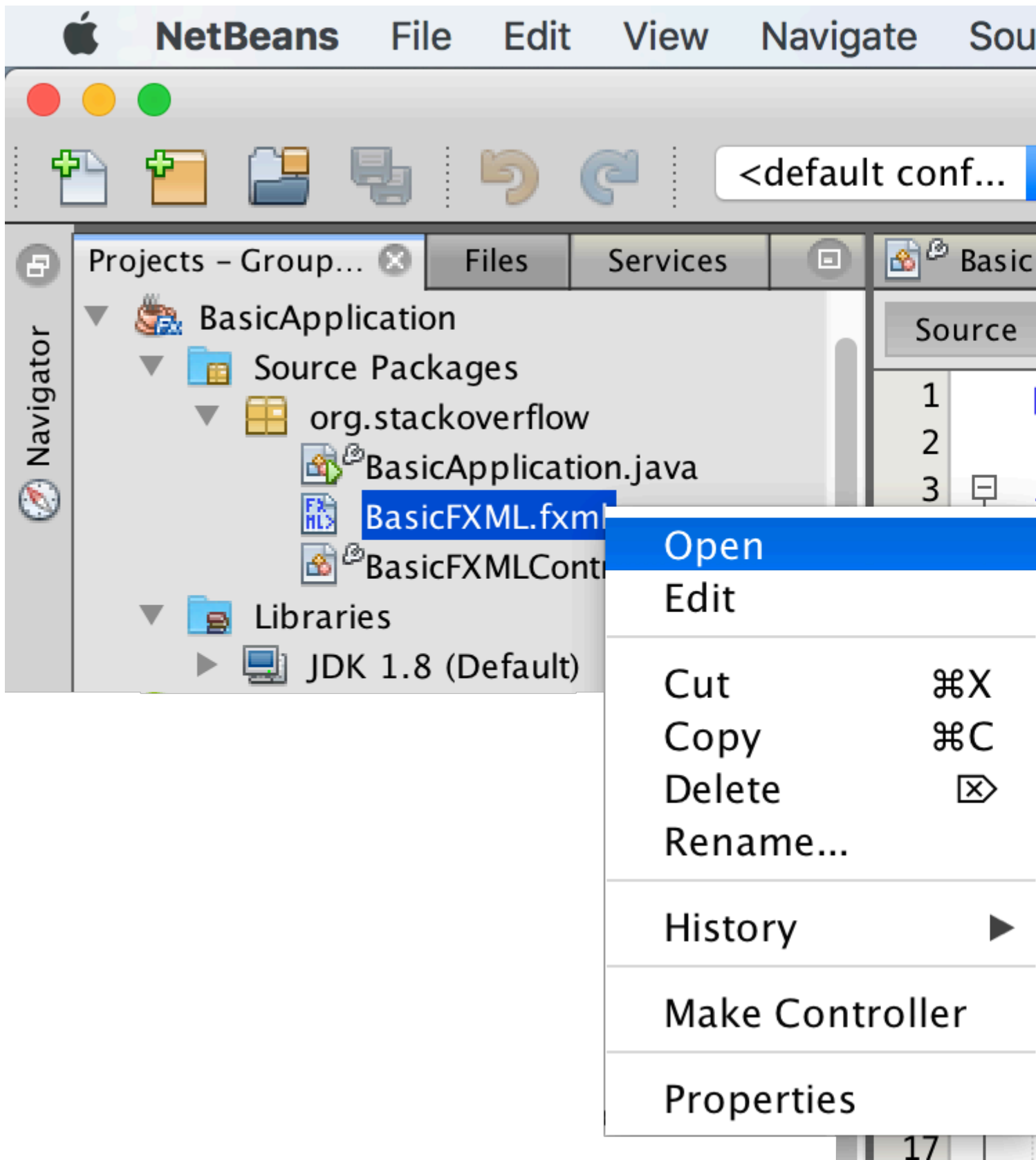
L'approccio migliore è aprire il file FXML con Scene Builder, in cui tutte le modifiche verranno salvate nel file.

È possibile avviare Scene Builder per aprire il file:

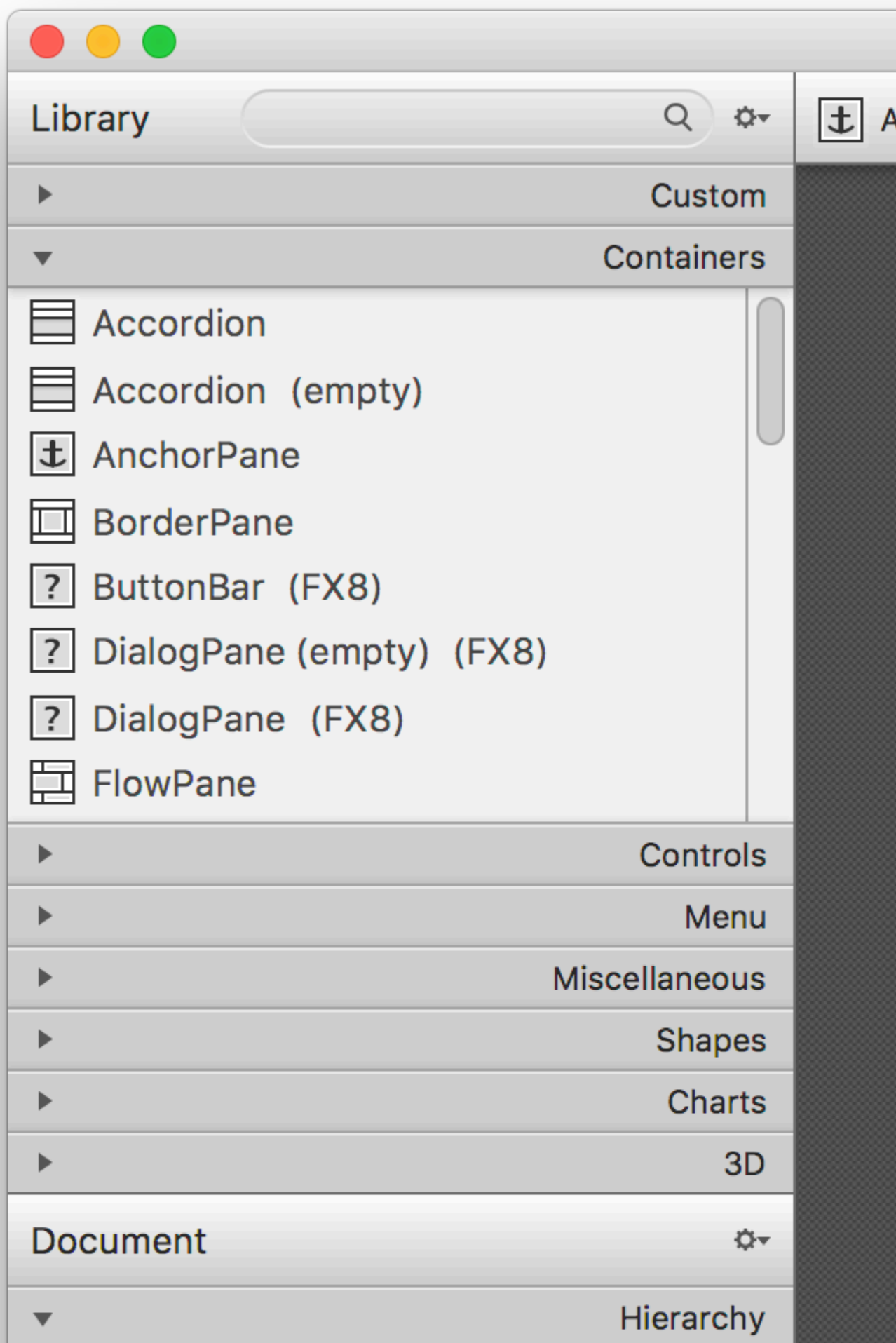


O il file può essere aperto con Scene Builder direttamente dall'IDE:

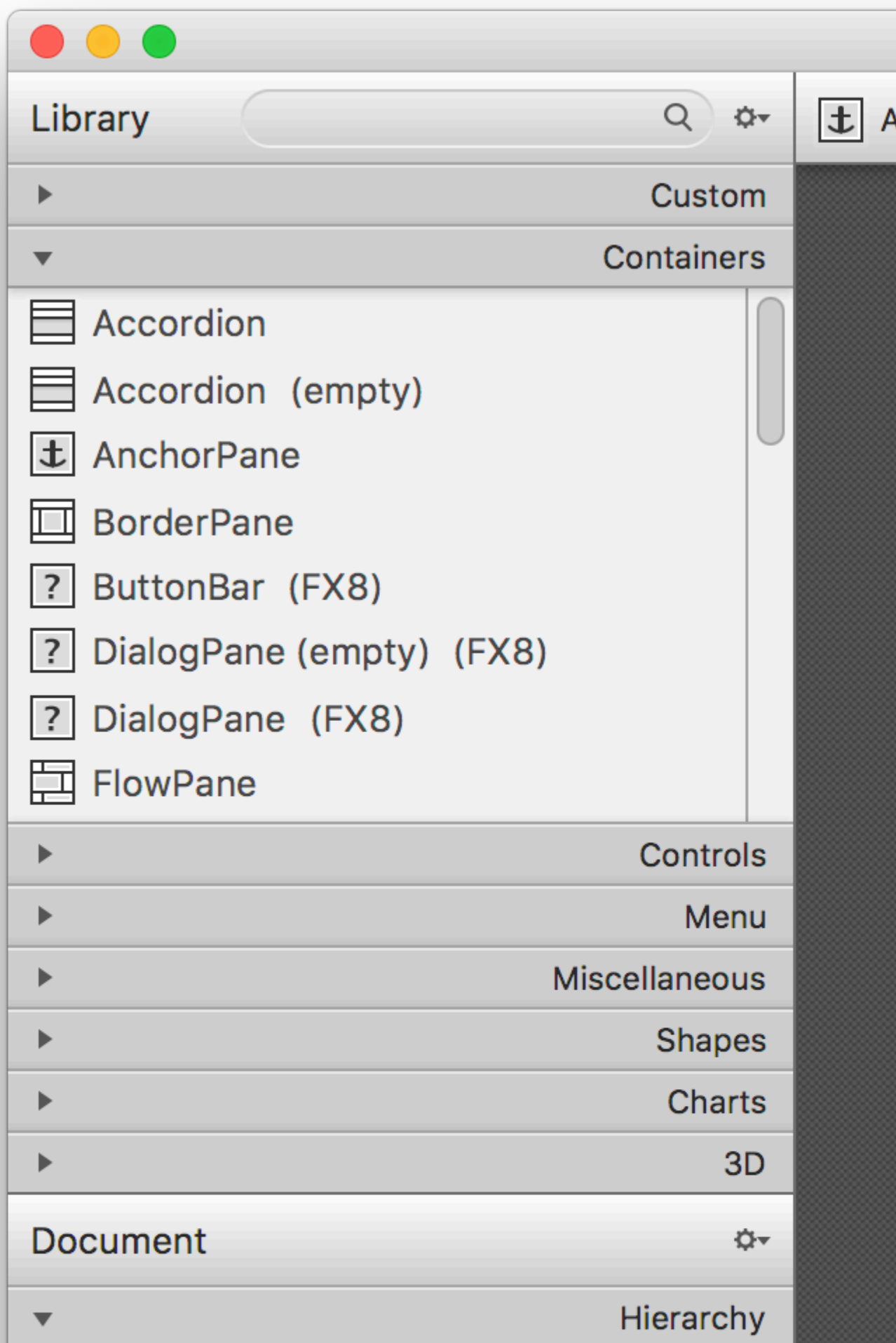
- Da NetBeans, nella scheda del progetto, fare doppio clic sul file o fare clic con il tasto destro e selezionare `Open`.
- Da IntelliJ, nella scheda del progetto, fai clic con il pulsante destro del mouse sul file e seleziona `Open In Scene Builder`.
- Da Eclipse, nella scheda del progetto, fare clic con il tasto destro sul file e selezionare `Open with Scene Builder`.



Se Scene Builder è installato correttamente e il suo percorso viene aggiunto all'IDE (vedi Note sotto), verrà aperto il file:



. Può essere impostato nel riquadro Code :



Capitolo 16: ScrollPane

introduzione

ScrollPane è un controllo che offre una visualizzazione dinamica del suo contenuto. Questa vista è controllata in vari modi; (pulsante di incremento / decremento del mouse) per avere una visione integrale del contenuto.

Examples

A) Corrette le dimensioni del contenuto:

La dimensione del contenuto sarà uguale a quella del contenitore ScrollPane.

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

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

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

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

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

B) Dimensioni del contenuto dinamico:

La dimensione del contenuto cambierà in base agli elementi aggiunti che superano i limiti di contenuto in entrambi gli assi (orizzontale e verticale) che possono essere visti spostandosi attraverso la vista.

```
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.
```

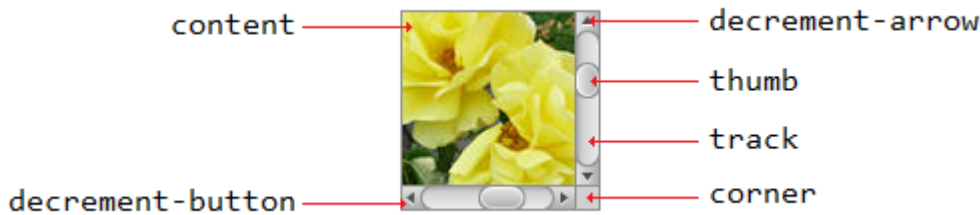
```
scrollpane.setContent(content); // we add the content to the ScrollPane
```

Nota: qui non abbiamo bisogno di entrambi i metodi (setFitToWidth / setFitToHeight).

Styling the ScrollPane:

L'aspetto di ScrollPane può essere facilmente modificato, avendo alcune nozioni di " CSS " e rispettando alcune *proprietà di "controllo"* e ovviamente avendo un po 'di " *immaginazione* ".

A) Gli elementi che compongono ScrollPane:



B) Proprietà CSS:

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

.scroll-bar:horizontal .track{}

.scroll-bar:horizontal .thumb{}

.scroll-bar:vertical .thumb{}

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

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

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

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

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

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

.scroll-pane .corner{}

.scroll-pane{}
```

Leggi ScrollPane online: <https://riptutorial.com/it/javafx/topic/8259/scrollpane>

Capitolo 17: Stampa

Examples

Stampa di base

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

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

Questo stampa sulla stampante predefinita senza mostrare alcuna finestra di dialogo all'utente. Per utilizzare una stampante diversa da quella predefinita, è possibile utilizzare `PrinterJob#createPrinterJob(Printer)` per impostare la stampante corrente. Puoi usarlo per vedere tutte le stampanti sul tuo sistema:

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

Stampa con finestra di dialogo del sistema

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

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

Leggi Stampa online: <https://riptutorial.com/it/javafx/topic/5157/stampa>

Capitolo 18: TableView

Examples

Sample TableView con 2 colonne

Articolo da tavola

La seguente classe contiene 2 proprietà un nome (`String`) e la dimensione (`double`). Entrambe le proprietà sono racchiuse in proprietà JavaFX per consentire a `TableView` di osservare le modifiche.

```
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;
    }

}
```

Applicazione di esempio

Questa applicazione mostra un `TableView` con 2 colonne; uno per il nome e uno per la dimensione

di una `Person`. Selezionando uno dei `Person`, i dati vengono aggiunti a `TextField` s sotto `TableView` e consentono all'utente di modificare i dati. Nota una volta che la modifica è stata `TableView`, `TableView` viene automaticamente aggiornato.

Per ogni per ogni `TableColumn` aggiunto al `TableView` un `cellValueFactory` viene assegnato. Questo factory è responsabile della conversione degli elementi tabella (`Person` s) in `ObservableValue` che contengono il valore che deve essere visualizzato nella cella della tabella e che consente a `TableView` di ascoltare le modifiche per questo valore.

```
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` può essere utilizzato come `cellValueFactory` in una `TableColumn`. Usa la reflection per accedere ai metodi che corrispondono a un determinato pattern per recuperare i dati da un oggetto `TableView`:

Esempio

```

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

```

Il nome del metodo utilizzato per ottenere i dati dipende dal parametro di configurazione del costruttore per `PropertyValueFactory`.

- **Metodo di proprietà:** questo tipo di metodo dovrebbe restituire un valore `ObservableValue` contenente i dati. Le modifiche possono essere osservate. Devono abbinare il `<constructor parameter>Property` e non prendere parametri.
- **Metodo Getter:** questo tipo di metodo si aspetta di restituire direttamente il valore (`String` nell'esempio sopra). Il nome del metodo deve corrispondere al modello `get<Constructor parameter>`. Nota che qui `<Constructor parameter>` inizia con una *lettera maiuscola*. Questo metodo non dovrebbe prendere parametri.

Esempi di nomi di metodi

| parametro costruttore (senza virgolette) | nome del metodo di proprietà | nome del metodo getter |
|--|------------------------------|------------------------|
| foo | fooProperty | getFoo |
| foobar | fooBarProperty | getFooBar |
| XYZ | XYZProperty | getXYZ |
| ListIndex | listIndexProperty | getListIndex |
| un valore | aValueProperty | getAValue |

Personalizzazione di TableCell in base all'elemento

A volte una colonna dovrebbe mostrare contenuti diversi rispetto al valore `toString` dell'oggetto cella. In questo caso, il `TableCell` s creato da `cellFactory` di `TableColumn` viene personalizzato per modificare il layout in base all'elemento.

Nota importante: `TableView` crea solo i `TableCell` mostrati nell'interfaccia utente. Gli oggetti all'interno delle celle possono cambiare e persino diventare vuoti. Il programmatore deve aver cura di annullare qualsiasi modifica a `TableCell` eseguita quando un elemento è stato aggiunto quando viene rimosso. In caso contrario, il contenuto potrebbe ancora essere visualizzato in una cella in cui "non appartiene".

Nell'esempio seguente, l'impostazione di un elemento determina il testo da impostare e l'immagine visualizzata in `ImageView` :

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

Se l'elemento diventa `null` o la cella diventa vuota, tali modifiche vengono annullate impostando i valori su `null` :

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

L'esempio seguente mostra un'emoji in aggiunta al testo in una `TableCell` .

Il metodo `updateItem` viene chiamato ogni volta che viene modificato l'elemento di una `Cell` . Sovrascrivere questo metodo consente di reagire alle modifiche e regolare l'aspetto della cella. Aggiungere un listener `itemProperty()` di una cella sarebbe un'alternativa, ma in molti casi `TableCell` viene esteso.

Tipo di elemento

```
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;
    }
}

```

Codice nella classe di applicazione

```

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);
    }
}

```

Classe cellulare

```

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());
        }
    }
}

```

Aggiungi pulsante a TableView

È possibile aggiungere un pulsante o un altro componente javafx a TableView utilizzando il `setCellFactory(Callback value) column setCellFactory(Callback value) .`

Applicazione di esempio

In questa applicazione stiamo per aggiungere un pulsante a TableView. Quando si fa clic sul pulsante di questa colonna, i dati sulla stessa riga del pulsante vengono selezionati e le informazioni vengono stampate.

Nel metodo `addButtonToTable()`, callback `cellFactory` è responsabile dell'aggiunta del pulsante alla colonna correlata. Definiamo la cella `cellFactory` chiamabile e implementiamo il suo metodo di `call(...)` override `call(...)` per ottenere `TableCell` con il pulsante e quindi questo `cellFactory` impostato sul metodo `setCellFactory(..)` colonna correlata. Nel nostro esempio questo è `colBtn.setCellFactory(cellFactory)`. SSCCE è qui sotto:

```
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;
}

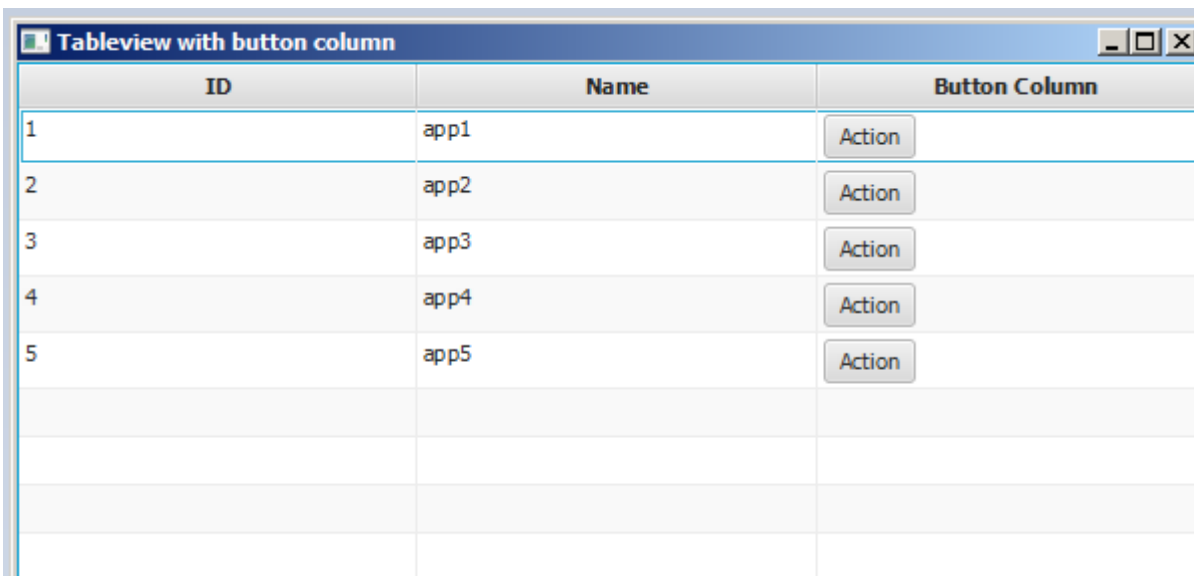
public String getName() {
    return name;
}

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

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

```

Immagine dello schermo:



| ID | Name | Button Column |
|----|------|---------------|
| 1 | app1 | Action |
| 2 | app2 | Action |
| 3 | app3 | Action |
| 4 | app4 | Action |
| 5 | app5 | Action |
| | | |
| | | |
| | | |

Leggi TableView online: <https://riptutorial.com/it/javafx/topic/2229/tableview>

Capitolo 19: Tela

introduzione

A `Canvas` è un `Node` JavaFX, rappresentato come un'area vuota e rettangolare, in grado di visualizzare immagini, forme e testo. Ogni `Canvas` contiene esattamente un oggetto `GraphicsContext`, responsabile della ricezione e del buffering delle chiamate `draw`, che, alla fine, vengono renderizzate sullo schermo da `Canvas`.

Examples

Forme di base

`GraphicsContext` fornisce una serie di metodi per disegnare e riempire forme geometriche. In genere, questi metodi richiedono che le coordinate vengano passate come parametri, direttamente o sotto forma di una matrice di valori `double`. Le coordinate sono sempre relative alla `Canvas`, la cui origine è nell'angolo in alto a sinistra.

Nota: `GraphicsContext` non si disegna al di fuori dei limiti del `Canvas`, ovvero tentare di disegnare all'esterno dell'area `Canvas` definita dalle sue dimensioni e ridimensionarlo in seguito non produrrà alcun risultato.

L'esempio seguente mostra come disegnare tre forme geometriche piene semitrasparenti delineate con un tratto nero.

```
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);
```



Leggi Tela online: <https://riptutorial.com/it/javafx/topic/8935/tela>

Capitolo 20: threading

Examples

Aggiornamento dell'interfaccia utente mediante Platform.runLater

Le operazioni a esecuzione prolungata non devono essere eseguite sul thread dell'applicazione JavaFX, poiché ciò impedisce a JavaFX di aggiornare l'interfaccia utente, determinando un'interfaccia utente bloccata.

Inoltre, qualsiasi modifica a un `Node` che fa parte di un grafico di scena "live" **deve** avvenire nel thread dell'applicazione JavaFX. `Platform.runLater` può essere utilizzato per eseguire tali aggiornamenti sul thread dell'applicazione JavaFX.

Nell'esempio seguente viene illustrato come aggiornare ripetutamente un `Node Text` da un thread diverso:

```
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);
}
}

```

Raggruppamento degli aggiornamenti dell'interfaccia utente

Il seguente codice rende l'interfaccia utente non risponde per un breve periodo dopo il clic del pulsante, poiché vengono utilizzate troppe chiamate `Platform.runLater`. (Prova a scorrere il `ListView` immediatamente dopo il clic del pulsante.)

```

@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();
}

```

Per evitare ciò invece di utilizzare un numero elevato di aggiornamenti, il seguente codice utilizza un `AnimationTimer` per eseguire l'aggiornamento solo una volta per frame:

```
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();
        }
    }
}

```

che consente di raggruppare gli aggiornamenti utilizzando la classe `Updater` :

```

private final Updater updater = new Updater();

...

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

```

Come utilizzare il servizio JavaFX

Invece di eseguire attività intensive in `JavaFX Thread` che dovrebbero essere eseguite in un `Service`. Quindi, in pratica, cos'è un [servizio](#) ?

Un servizio è una classe che sta creando un nuovo `Thread` ogni volta che si avvia esso ed è

passato un **task** ad esso per fare un po 'Servizio work.The può restituire o meno un valore.

Di seguito è riportato un tipico esempio di servizio JavaFX che sta facendo un po 'di lavoro e restituisce una `Map<String, String>()`:

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

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

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

        // if failed
        setOnFailed(fail -> {
            //code if 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;
}
};
}
}
```

Leggi threading online: <https://riptutorial.com/it/javafx/topic/2230/threading>

Capitolo 21: WebView e WebEngine

Osservazioni

`WebView` è il nodo JavaFX che è integrato nell'albero dei componenti JavaFX. Gestisce un `WebEngine` e ne mostra il contenuto.

Il `WebEngine` è il motore di `WebEngine` sottostante, che fondamentalemente fa tutto il lavoro.

Examples

Caricamento di una pagina

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

`WebView` è la shell dell'interfaccia utente di `WebEngine`. Quasi tutti i controlli per l'interazione non dell'interfaccia utente con una pagina vengono eseguiti tramite la classe `WebEngine`.

Ottieni la cronologia delle pagine di una WebView

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

La cronologia è fondamentalemente una lista di voci. Ogni voce rappresenta una pagina visitata e fornisce l'accesso alle informazioni sulla pagina rilevanti, come l'URL, il titolo e la data di ultima visita della pagina.

L'elenco può essere ottenuto utilizzando il metodo `getEntries()`. La cronologia e l'elenco di voci corrispondente cambiano man `WebEngine` che `WebEngine` naviga sul Web. L'elenco può espandersi o restringersi in base alle azioni del browser. Queste modifiche possono essere ascoltate dall'API `ObservableList` che l'elenco espone.

L'indice della voce della cronologia associata alla pagina attualmente visitata è rappresentato da `currentIndexProperty()`. L'indice corrente può essere utilizzato per navigare a qualsiasi voce nella cronologia utilizzando il metodo `go(int)`. `maxSizeProperty()` imposta la dimensione massima della cronologia, che è la dimensione dell'elenco cronologico.

Di seguito è riportato un esempio di come [ottenere ed elaborare l'elenco degli elementi della cronologia Web](#).

Un `ComboBox` (`comboBox`) viene utilizzato per memorizzare gli elementi della cronologia. Usando `ListChangeListener` su `WebHistory` il `ComboBox` viene aggiornato alla `WebHistory` corrente. Sul `ComboBox` è un `EventHandler` che reindirizza alla pagina selezionata.

```

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();
    }
});

```

inviare avvisi JavaScript dalla pagina Web visualizzata al log delle applicazioni Java.

```

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

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

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

```

Comunicazione tra app Java e Javascript nella pagina web

Quando si utilizza una WebView per visualizzare la propria pagina Web personalizzata e questa pagina Web contiene Javascript, potrebbe essere necessario stabilire una comunicazione bidirezionale tra il programma Java e Javascript nella pagina Web.

Questo esempio mostra come impostare una tale comunicazione.

La pagina web deve visualizzare un campo di immissione e un pulsante. Facendo clic sul pulsante, il valore dal campo di input viene inviato all'applicazione Java, che lo elabora. Dopo l'elaborazione, un risultato viene inviato al Javascript che a sua volta visualizza il risultato sulla pagina web.

Il principio di base è che per la comunicazione da Javascript a Java viene creato un oggetto in Java che viene impostato nella pagina web. E per l'altra direzione, un oggetto viene creato in Javascript ed estratto dalla pagina web.

Il codice seguente mostra la parte Java, ho tenuto tutto in un unico file:

```
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());
        }
    }
}
}

```

Una volta caricata la pagina, un oggetto `JavaConnector` (definito dalla classe interna e creato come campo) viene impostato nella pagina Web da tali chiamate:

```

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

```

L'oggetto `javascriptConnector` viene recuperato dalla pagina Web con

```

javascriptConnector = (JavaScript) webEngine.executeScript("getJsConnector()");

```

Quando viene chiamato il metodo `toLowerCase(String)` da `JavaConnector`, il valore passato viene convertito e quindi restituito tramite l'oggetto `javascriptConnector`.

E questo è il codice html e javascript:

```

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

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

    </main>

    <script type="text/javascript">
      function sendToJava () {
        var s = document.getElementById('input').value;
        javaConnector.toLowerCase(s);
      };
    </script>
  </body>
</html>

```

```

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

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

```

La funzione `sendToJava` chiama il metodo del `JavaConnector` impostato dal codice Java:

```

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

```

e la funzione chiamata dal codice Java per recuperare `javascriptConnector` restituisce semplicemente l'oggetto `jsConnector` :

```

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

function getJsConnector() {
    return jsConnector;
};

```

Il tipo di argomento delle chiamate tra Java e Javascript non è limitato a Stringhe. Ulteriori informazioni sui possibili tipi e conversioni si trovano nel [documento dell'API JSObject](#) .

Leggi [WebView e WebEngine online](#): <https://riptutorial.com/it/javafx/topic/5156/webview-e-webengine>

Titoli di coda

| S. No | Capitoli | Contributors |
|-------|----------------------------------|--|
| 1 | Iniziare con javafx | Community , CraftedCart , D3181 , DVarga , fabian , Ganesh , Hendrik Ebbers , Petter Friberg |
| 2 | Animazione | fabian , J Atkin |
| 3 | Binding JavaFX | Alexiy |
| 4 | CSS | fabian |
| 5 | finestre | fabian , GtknBtn |
| 6 | Finestre di dialogo | fabian , GtknBtn , Modus Tollens |
| 7 | FXML e controller | D3181 , fabian , James_D |
| 8 | Grafico | Dth , James_D , Jinu P C |
| 9 | Internazionalizzazione in JavaFX | ItachiUchiha , Joffrey , Nico T , P.J.Meisch |
| 10 | layout | DVarga , fabian , Filip Smola , Jinu P C , Sohan Chowdhury , trashgod |
| 11 | paginatura | fabian , J Atkin |
| 12 | Proprietà e osservabili | fabian |
| 13 | Pulsante | Dth , DVarga , J Atkin , Maverick283 , Nico T , Squidward |
| 14 | Pulsante radio | Nico T |
| 15 | Scene Builder | Ashlyn Campbell , José Pereda |
| 16 | ScrollPane | Bo Halim |
| 17 | Stampa | J Atkin , Squidward |
| 18 | TableView | Bo Halim , fabian , GtknBtn |
| 19 | Tela | Dth |
| 20 | threading | Brendan , fabian , GOXR3PLUS , James_D , Koko Essam , sazzy4o |

| | | |
|----|------------------------|---|
| 21 | WebView e WebEngine | fabian , J Atkin , James_D , P.J.Meisch , Squidward |
|----|------------------------|---|