



EBook Gratis

APRENDIZAJE javafx

Free unaffiliated eBook created from
Stack Overflow contributors.

#javafx

Tabla de contenido

Acerca de	1
Capítulo 1: Empezando con javafx	2
Observaciones.....	2
Versiones.....	2
Examples.....	2
Instalación o configuración.....	2
Programa hola mundo.....	3
Capítulo 2: Animación	5
Examples.....	5
Animando una propiedad con línea de tiempo.....	5
Capítulo 3: Botón	6
Examples.....	6
Añadiendo un oyente de acción.....	6
Añadiendo un gráfico a un botón.....	6
Crear un botón.....	6
Predeterminado y botones de cancelación.....	7
Capítulo 4: Boton de radio	8
Examples.....	8
Creando botones de radio.....	8
Usar grupos en los botones de radio.....	8
Eventos para botones de radio.....	8
Solicitando enfoque para botones de radio.....	9
Capítulo 5: Constructor de escena	10
Introducción.....	10
Observaciones.....	10
Instalación de Scene Builder.....	10
Un poquito de historia.....	13
Tutoriales.....	14
Controles personalizados.....	14
SO preguntas.....	15

Examples.....	15
Proyecto JavaFX básico usando FXML.....	15
Capítulo 6: CSS.....	20
Sintaxis.....	20
Examples.....	20
Usando CSS para el estilo.....	20
Rectángulo extensible que agrega nuevas propiedades stylable.....	22
Nodo personalizado.....	22
Capítulo 7: Diálogos.....	26
Observaciones.....	26
Examples.....	26
TextInputDialog.....	26
ChoiceDialog.....	26
Alerta.....	27
Ejemplo.....	27
Texto de botón personalizado.....	27
Capítulo 8: Diseños.....	28
Examples.....	28
StackPane.....	28
HBox y VBox.....	28
BorderPane.....	30
FlowPane.....	31
GridPane.....	33
Hijos del GridPane.....	33
Añadiendo niños al GridPane.....	33
Tamaño de columnas y filas.....	34
Alineación de elementos dentro de las celdas de la cuadrícula.....	35
TilePane.....	35
AnchorPane.....	36
Capítulo 9: Enhebrado.....	39
Examples.....	39
Actualizando la interfaz de usuario usando Platform.runLater.....	39

Agrupar las actualizaciones de la interfaz de usuario.....	40
Cómo utilizar el servicio JavaFX.....	41
Capítulo 10: Enlaces JavaFX.....	44
Examples.....	44
Enlace de propiedad simple.....	44
Capítulo 11: FXML y controladores.....	45
Sintaxis.....	45
Examples.....	45
Ejemplo FXML.....	45
Controladores anidados.....	47
Definir bloques y.....	49
Pasando datos a FXML - accediendo al controlador existente.....	50
Pasando datos a FXML - Especificando la instancia del controlador.....	51
Pasando parámetros a FXML - usando un controllerFactory.....	52
Creación de instancias en FXML.....	54
Una nota sobre las importaciones.....	55
@NamedArg anotado @NamedArg.....	55
No args constructor.....	56
fx:value atributo de fx:value.....	56
fx:factory.....	56
<fx:copy>.....	57
fx:constant.....	57
Configuración de propiedades.....	57
etiqueta <property>.....	57
Propiedad por defecto.....	58
property="value" atributo property="value".....	58
colocadores estáticos.....	58
Tipo de coerción.....	58
Ejemplo.....	59
Capítulo 12: Gráfico.....	63
Examples.....	63

Gráfico circular	63
Constructores	63
Datos	63
Ejemplo	63
Salida:	64
Gráfico circular interactivo	65
Gráfico de linea	65
Ejes	66
Ejemplo	66
Salida:	67
Capítulo 13: Impresión	68
Examples	68
Impresión básica	68
Impresión con diálogo del sistema	68
Capítulo 14: Internacionalización en JavaFX	69
Examples	69
Cargando paquete de recursos	69
Controlador	69
Cambio dinámico de idioma cuando la aplicación se está ejecutando	69
Capítulo 15: Lona	75
Introducción	75
Examples	75
Formas básicas	75
Capítulo 16: Paginación	77
Examples	77
Creando una paginación	77
Avance automático	77
Cómo funciona	77
Crear una paginación de imágenes	78
Cómo funciona	78
Capítulo 17: Propiedades y observables	79

Observaciones.....	79
Examples.....	79
Tipos de propiedades y nombres.....	79
Propiedades estandar.....	79
Propiedades de la lista de solo lectura.....	79
Propiedades de mapas de solo lectura.....	79
Ejemplo de StringProperty.....	80
Ejemplo de ReadOnlyIntegerProperty.....	80
Capítulo 18: ScrollPane.....	83
Introducción.....	83
Examples.....	83
A) Tamaño del contenido fijo:.....	83
B) Tamaño del contenido dinámico:.....	83
Diseñando el ScrollPane:.....	84
Capítulo 19: TableView.....	85
Examples.....	85
Ejemplo TableView con 2 columnas.....	85
PropertyValueFactory.....	88
Personalizar el aspecto de TableCell dependiendo del artículo.....	89
Añadir botón a Tableview.....	92
Capítulo 20: WebView y WebEngine.....	96
Observaciones.....	96
Examples.....	96
Cargando una pagina.....	96
Obtener el historial de la página de un WebView.....	96
envíe alertas de Javascript desde la página web mostrada al registro de aplicaciones Java.....	97
Comunicación entre la aplicación Java y Javascript en la página web.....	97
Capítulo 21: Windows.....	101
Examples.....	101
Creando una nueva ventana.....	101
Creación de un diálogo personalizado.....	101
Creación de un diálogo personalizado.....	106

Acerca de

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

Capítulo 1: Empezando con javafx

Observaciones

JavaFX es una plataforma de software para crear y entregar aplicaciones de escritorio, así como aplicaciones de Internet enriquecidas (RIA) que pueden ejecutarse en una amplia variedad de dispositivos. JavaFX está destinado a reemplazar Swing como la biblioteca de GUI estándar para Java SE.

TI permite a los desarrolladores diseñar, crear, probar, depurar e implementar aplicaciones de cliente enriquecidas.

La apariencia de las aplicaciones JavaFX se puede personalizar utilizando Hojas de estilo en cascada (CSS) para el estilo (ver [JavaFX: CSS](#)) y (F) Los archivos XML se pueden usar para estructurar objetos, lo que facilita la creación o el desarrollo de una aplicación (consulte [FXML y controladores](#)). Scene Builder es un editor visual que permite la creación de archivos fxml para una UI sin escribir código.

Versiones

Versión	Fecha de lanzamiento
JavaFX 2	2011-10-10
JavaFX 8	2014-03-18

Examples

Instalación o configuración

Las API de JavaFX están disponibles como una función totalmente integrada del Java SE Runtime Environment (JRE) y el Java Development Kit (JDK). Debido a que el JDK está disponible para todas las plataformas de escritorio principales (Windows, Mac OS X y Linux), las aplicaciones JavaFX compiladas para JDK 7 y posteriores también se ejecutan en todas las plataformas de escritorio principales. El soporte para plataformas ARM también se ha hecho disponible con JavaFX 8. JDK para ARM incluye los componentes de base, gráficos y controles de JavaFX.

Para instalar JavaFX, instale la versión elegida del entorno de Java Runtime y el [kit de desarrollo de Java](#).

Las características ofrecidas por JavaFX incluyen:

1. API de Java.
2. FXML y Scene Builder.

3. WebView.
4. Interoperabilidad del swing.
5. Controles de UI incorporados y CSS.
6. Tema de modena
7. Características de gráficos 3D.
8. API de lienzo.
9. API de impresión.
10. Soporte de texto enriquecido.
11. Soporte multitáctil.
12. Soporte Hi-DPI.
13. Tubería gráfica acelerada por hardware.
14. Motor multimedia de alto rendimiento.
15. Modelo de despliegue de aplicaciones autónomas.

Programa hola mundo

El siguiente código crea una interfaz de usuario simple que contiene un solo `Button` que imprime una `String` en la consola al hacer 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 clase de `Application` es el punto de entrada de cada aplicación JavaFX. Solo se puede iniciar una `Application` y esto se hace usando

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

Esto crea una instancia de la clase de `Application` pasada como parámetro e inicia la plataforma JavaFX.

Lo siguiente es importante para el programador aquí:

1. El primer `launch` crea una nueva instancia de la clase de `Application` (`HelloWorld` en este caso). La clase de `Application`, por lo tanto, necesita un constructor sin argumentos.
2. Se llama a `init()` en la instancia de la `Application` creada. En este caso, la implementación predeterminada de la `Application` no hace nada.
3. Se llama a `start` para la instancia de `Application` y la `Stage` primaria (= ventana) se pasa al método. Este método se llama automáticamente en el subproceso de la aplicación JavaFX (subproceso de plataforma).
4. La aplicación se ejecuta hasta que la plataforma determina que es hora de apagarse. Esto se hace cuando se cierra la última ventana en este caso.
5. El método de `stop` se invoca en la instancia de la `Application`. En este caso la implementación desde la `Application` no hace nada. Este método se llama automáticamente en el subproceso de la aplicación JavaFX (subproceso de plataforma).

En el método de `start` se construye el gráfico de escena. En este caso contiene 2 `Node` s: un `Button` y un `StackPane`.

El `Button` representa un botón en la interfaz de usuario y el `StackPane` es un contenedor para el `Button` que determina su ubicación.

Se crea una `Scene` para mostrar estos `Node`. Finalmente, la `Scene` se agrega al `Stage` que es la ventana que muestra la IU completa.

Lea Empezando con javafx en línea: <https://riptutorial.com/es/javafx/topic/887/empezando-con-javafx>

Capítulo 2: Animación

Examples

Animando una propiedad con línea de tiempo

```
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();
```

La forma más básica y flexible de usar la animación en JavaFX es con la clase `Timeline`. Una línea de tiempo funciona utilizando `KeyFrame` s como puntos conocidos en la animación. En este caso se sabe que al comienzo (0 seconds) que el `translateXProperty` tiene que ser cero, y al final (2 seconds) que la propiedad tiene que ser 80 . También puede hacer otras cosas, como configurar la animación para que se revierta, y cuántas veces debe ejecutarse.

Las líneas de tiempo pueden animar múltiples propiedades al mismo tiempo:

```
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
```

Esta animación tomará la propiedad `Y` de 0 (valor inicial de la propiedad) a 10 un segundo, y termina en 90 a los tres segundos. Tenga en cuenta que cuando la animación comienza de nuevo, `Y` vuelve a cero, aunque no sea el primer valor en la línea de tiempo.

Lea Animación en línea: <https://riptutorial.com/es/javafx/topic/5166/animacion>

Capítulo 3: Botón

Examples

Añadiendo un oyente de acción

Los botones activan los eventos de acción cuando se activan (p. Ej., Se hace clic en una combinación de teclas para el botón, ...).

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

Si está utilizando Java 8+, puede usar lambdas para escuchas de acción.

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

Añadiendo un gráfico a un botón.

Los botones pueden tener un gráfico. `graphic` puede ser cualquier nodo JavaFX, como un `ProgressBar`

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

Un `ImageView`

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

O incluso otro botón

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

Crear un botón

La creación de un `Button` es simple:

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

Esto creará un nuevo `Button` sin ningún texto o gráfico dentro.

Si desea crear un `Button` con un texto, simplemente use el constructor que toma una `String` como

parámetro (que establece la `textProperty` de texto del `Button`):

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

Si desea crear un `Button` con un gráfico en el interior o en cualquier otro `Node`, use este constructor:

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

Predeterminado y botones de cancelación

La API de `Button` proporciona una manera fácil de asignar atajos de teclado comunes a los botones sin la necesidad de acceder a la lista de aceleradores asignados a la `Scene` o escuchar explícitamente los eventos clave. A saber, se proporcionan dos métodos de conveniencia:

`setDefaultButton` y `setCancelButton`:

- Si configura `setDefaultButton` en `true`, el `Button` se `KeyCode.ENTER` cada vez que reciba un evento `KeyCode.ENTER`.
- Establecer `setCancelButton` en `true` hará que el `Button` dispare cada vez que reciba un evento `KeyCode.ESCAPE`.

El siguiente ejemplo crea una `Scene` con dos botones que se activan cuando se presionan las teclas de entrada o de escape, independientemente de si están enfocadas o no.

```
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);
```

El código anterior no funcionará si estas `KeyEvents` son consumidos por cualquier padre `Node`:

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

Lea Botón en línea: <https://riptutorial.com/es/javafx/topic/5162/boton>

Capítulo 4: Boton de radio

Examples

Creando botones de radio

Los botones de radio le permiten al usuario elegir un elemento de los dados. Hay dos formas de declarar un `RadioButton` con un texto aparte. Usando el constructor predeterminado `RadioButton()` y configurando el texto con el método `setText(String)` o usando el otro constructor

`RadioButton(String)` .

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

Como `RadioButton` es una extensión de `Labeled` también puede haber una `Image` especificada para `RadioButton` . Después de crear el `RadioButton` con uno de los constructores, simplemente agregue la `Image` con el `setGraphic(ImageView)` como aquí:

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

Usar grupos en los botones de radio

A `ToggleGroup` se utiliza para gestionar la `RadioButton` s de modo que sólo uno de cada grupo se pueden seleccionar en cada momento.

Crea un `ToggleGroup` simple como el siguiente:

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

Después de crear un `ToggleGroup` se puede asignar a `RadioButton` s usando `setToggleGroup(ToggleGroup)` . Use `setSelected(Boolean)` para preseleccionar uno de los `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);
```

Eventos para botones de radio

Normalmente, cuando se selecciona uno de los `RadioButton` s en un `ToggleGroup` la aplicación realiza una acción. A continuación se muestra un ejemplo que imprime los datos de usuario del `RadioButton` seleccionado que se ha configurado 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());
    }
});
```

Solicitando enfoque para botones de radio

Digamos que el segundo `RadioButton` cada tres está preseleccionado con `setSelected(Boolean)` , el enfoque sigue siendo el primer `RadioButton` por defecto. Para cambiar esto use el método `requestFocus()` .

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

Lea Boton de radio en línea: <https://riptutorial.com/es/javafx/topic/5906/boton-de-radio>

Capítulo 5: Constructor de escena

Introducción

JavaFX Scene Builder es una herramienta de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario de aplicaciones JavaFX, sin codificación. Se utiliza para generar archivos FXML.

Observaciones

JavaFX Scene Builder es una herramienta de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario de aplicaciones JavaFX, sin codificación. Los usuarios pueden arrastrar y soltar los componentes de la interfaz de usuario a un área de trabajo, modificar sus propiedades, aplicar hojas de estilo y el código FXML para el diseño que están creando se genera automáticamente en el fondo. El resultado es un archivo FXML que luego puede combinarse con un proyecto Java vinculando la interfaz de usuario a la lógica de la aplicación.

Desde una perspectiva de Model View Controller (MVC):

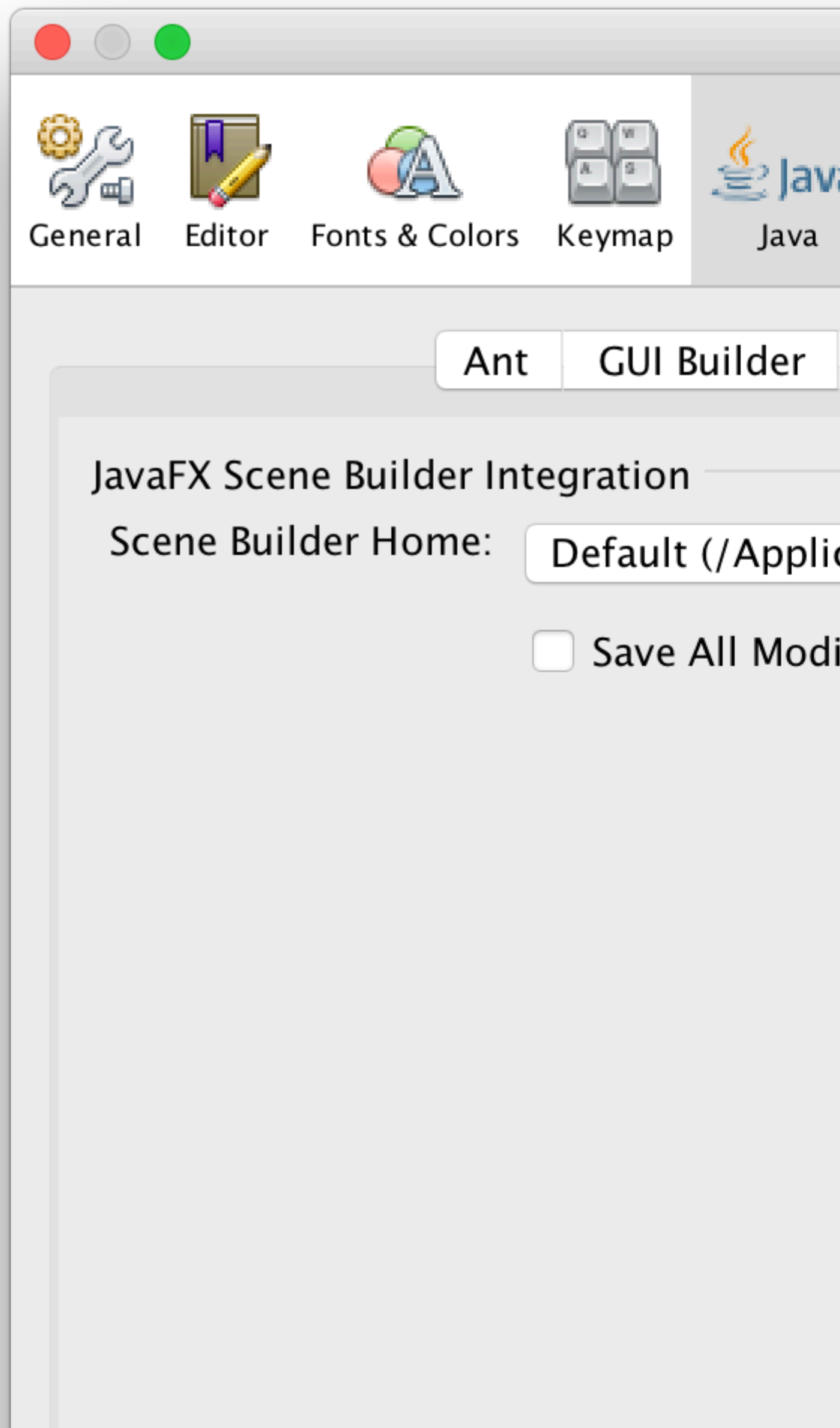
- El archivo FXML, que contiene la descripción de la interfaz de usuario, es la vista.
- El controlador es una clase Java, que implementa opcionalmente la clase Initializable, que se declara como el controlador para el archivo FXML.
- El modelo consta de objetos de dominio, definidos en el lado de Java, que se pueden conectar a la vista a través del controlador.

Instalación de Scene Builder

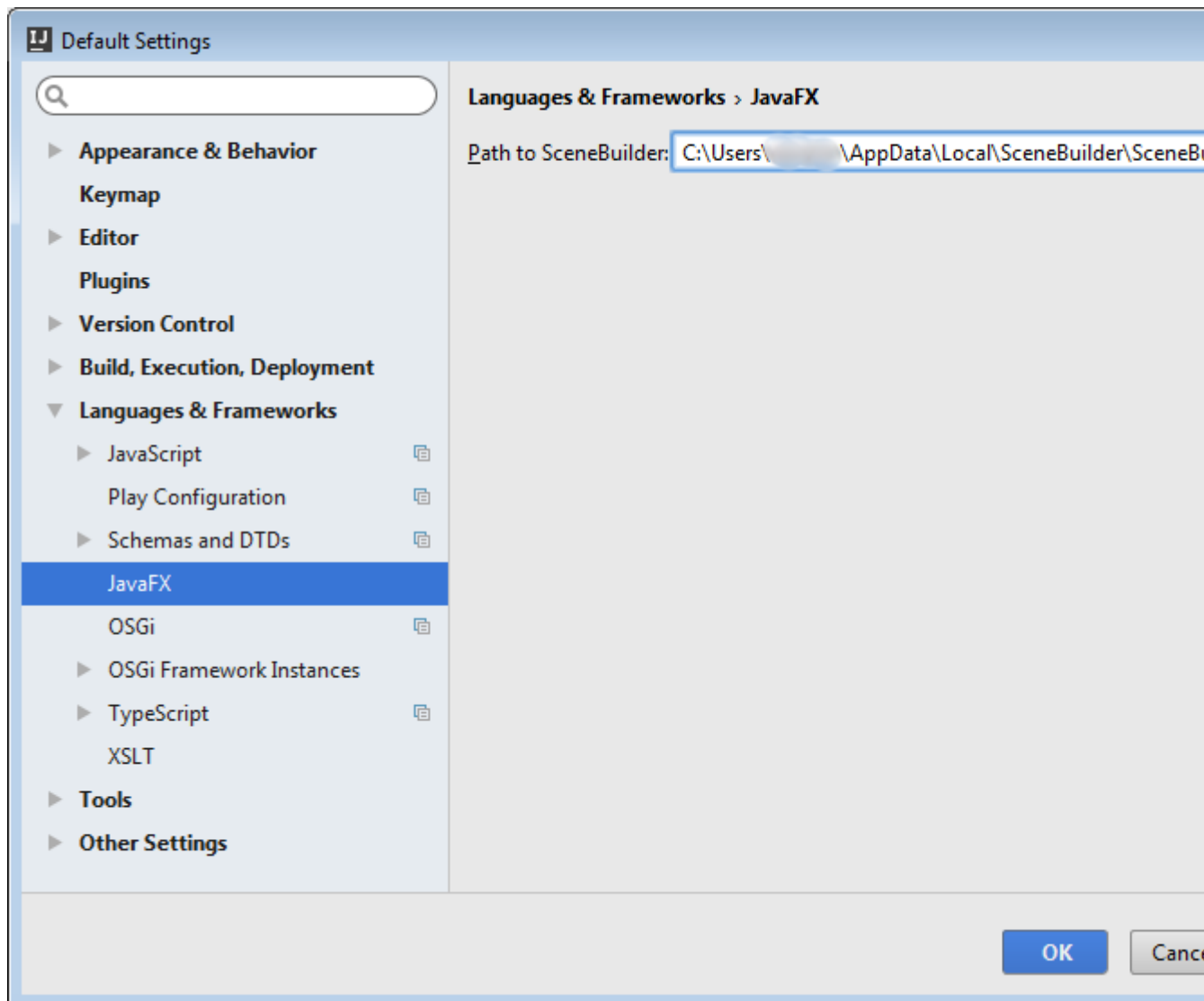
1. Descargue la versión más reciente de Scene Builder del [sitio web](#) de Gluon, seleccionando el instalador para su plataforma o el archivo ejecutable.
2. Con el instalador descargado, haga doble clic para instalar Scene Builder en su sistema. Se incluye un JRE actualizado.
3. Haga doble clic en el icono de Scene Builder para ejecutarlo como una aplicación independiente.
4. Integración IDE

Si bien Scene Builder es una aplicación independiente, produce archivos FXML que se integran con un proyecto Java SE. Al crear este proyecto en un IDE, es conveniente incluir un enlace a la ruta de Scene Builder, para poder editar los archivos FXML.

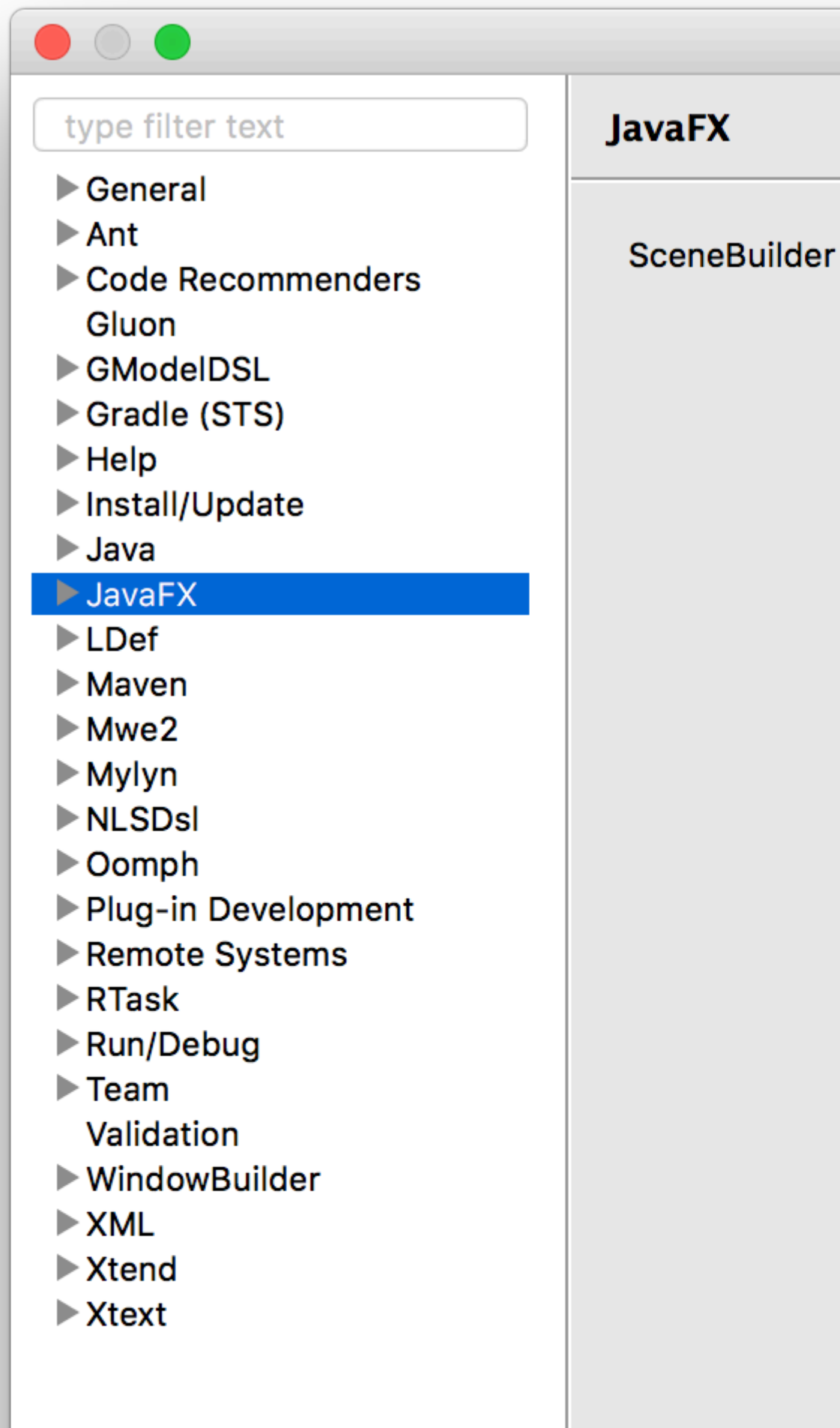
- NetBeans: en Windows vaya a NetBeans-> Herramientas-> Opciones-> Java-> JavaFX. En Mac OS X vaya a NetBeans-> Preferencias-> Java-> JavaFX. Proporcione el camino para el Hogar de creadores de escenas.



- IntelliJ: En Windows, vaya a IntelliJ-> Configuración-> Idiomas y marcos-> JavaFX. En Mac OS X vaya a IntelliJ-> Preferencias-> Idiomas y marcos-> JavaFX. Proporcione el camino para el Hogar de creadores de escenas.



- Eclipse: en Windows vaya a Eclipse-> Ventana-> Preferencias-> JavaFX. En Mac OS X vaya a Eclipse-> Preferencias-> JavaFX. Proporcione el camino para el Hogar de creadores de escenas.



binarios, hasta Scene Builder v 2.0, que incluía solo las funciones de JavaFX antes del lanzamiento de Java SE 8u40, por lo que no se incluyen las nuevas funciones como los controles `Spinner` .

[Gluon se](#) hizo cargo de la distribución de versiones binarias, y se puede descargar un Scene Builder 8+ actualizado para cada plataforma desde [aquí](#) .

Incluye los últimos cambios en JavaFX y también las mejoras y correcciones de errores recientes.

El proyecto de código abierto se puede encontrar [aquí](#) donde se pueden crear problemas, solicitudes de características y solicitudes de extracción.

Los archivos binarios heredados de Oracle todavía se pueden descargar desde [aquí](#) .

Tutoriales

Los tutoriales de Scene Builder se pueden encontrar aquí:

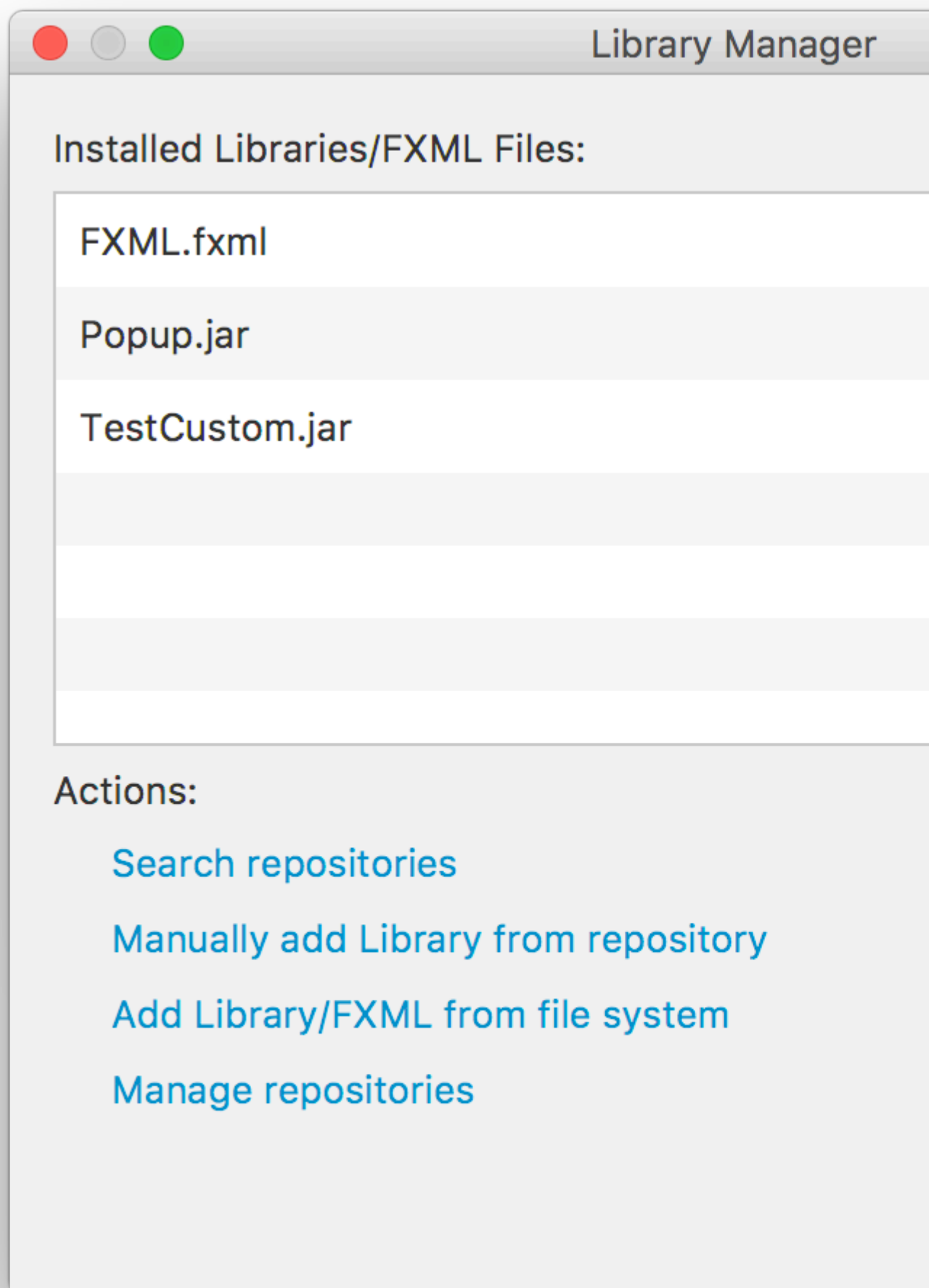
- [Tutorial de](#) Oracle Scene Builder 2.0

Los tutoriales de FXML se pueden encontrar aquí.

- [Tutorial de](#) Oracle FXML

Controles personalizados

Gluon ha [documentado](#) completamente la nueva función que permite importar frascos de terceros con controles personalizados, utilizando el Administrador de biblioteca (disponible desde Scene Builder 8.2.0).



desde el jar / classpath, según lo especificado por

```
FXMLLoader.load(getClass().getResource("BasicFXML.fxml")) .
```

Al cargar `basicFXML.fxml` , el cargador encontrará el nombre de la clase del controlador, como lo especifica `fx:controller="org.stackoverflow.BasicFXMLController"` en el FXML.

Luego, el cargador creará una instancia de esa clase, en la que intentará inyectar todos los objetos que tienen un `fx:id` en el FXML y están marcados con la anotación `@FXML` en la clase del controlador.

En este ejemplo, el `FXMLLoader` creará la etiqueta basada en `<Label ... fx:id="label"/>` , e inyectará la instancia de la etiqueta en la `@FXML private Label label;` .

Finalmente, cuando se haya cargado todo el FXML, `FXMLLoader` llamará al método de `initialize` del controlador, y se ejecutará el código que registra un controlador de acción con el botón.

Edición

Si bien el archivo FXML se puede editar dentro del IDE, no se recomienda, ya que el IDE proporciona solo la comprobación de sintaxis básica y el autocompletado, pero no la guía visual.

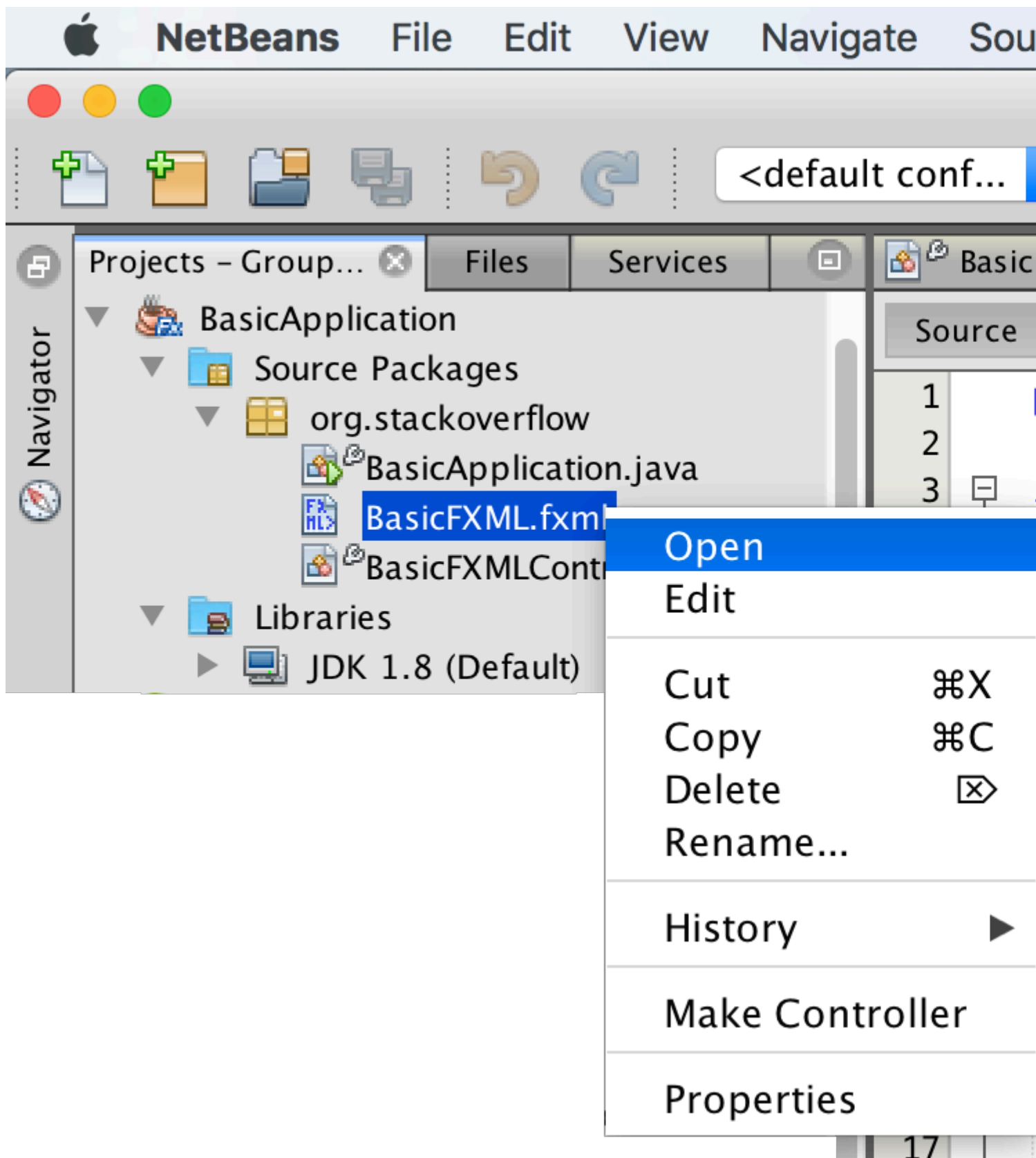
El mejor enfoque es abrir el archivo FXML con Scene Builder, donde todos los cambios se guardarán en el archivo.

Se puede iniciar Scene Builder para abrir el archivo:

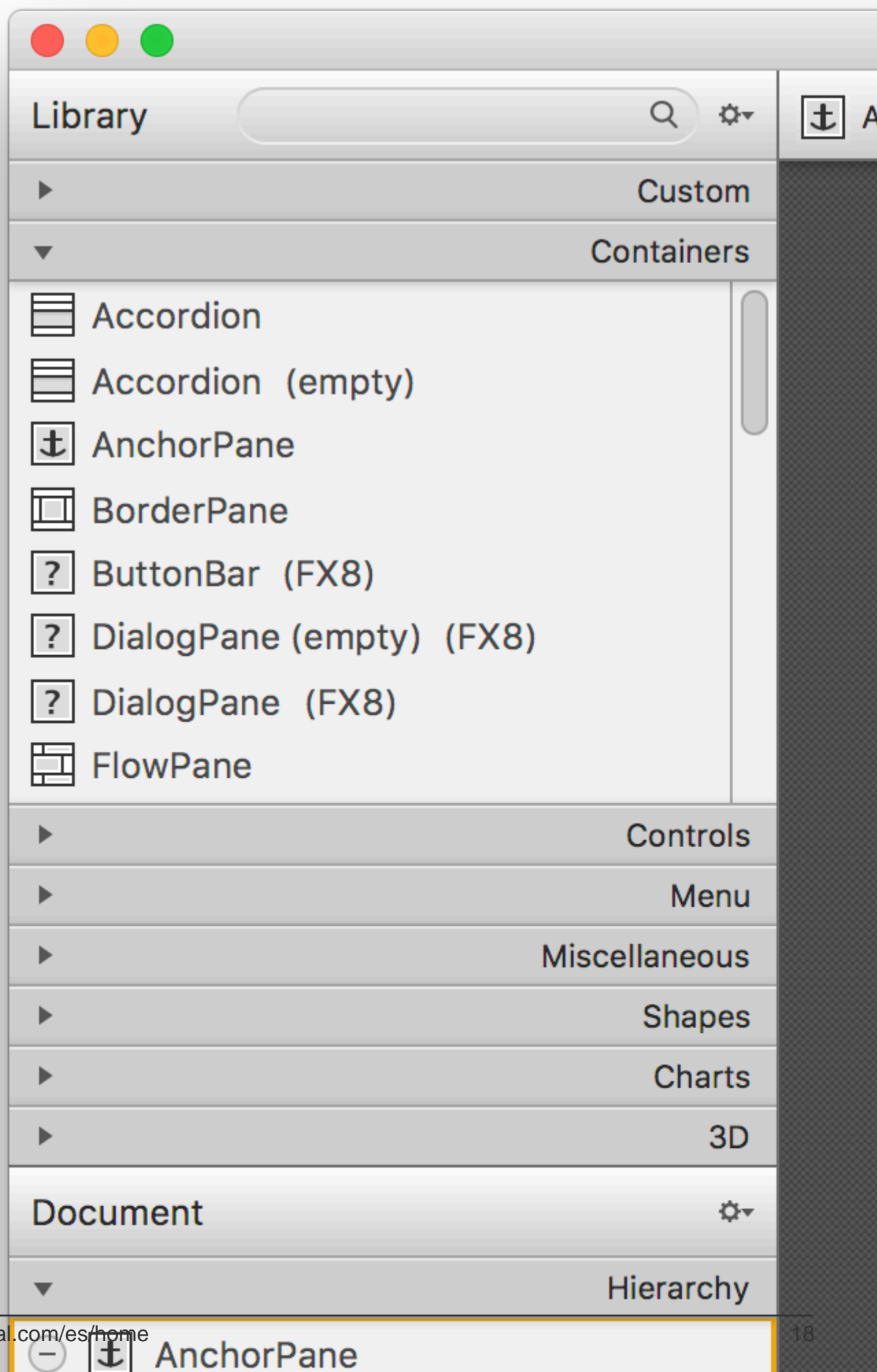


O el archivo se puede abrir con Scene Builder directamente desde el IDE:

- Desde NetBeans, en la pestaña del proyecto, haga doble clic en el archivo o haga clic derecho y seleccione `Open` .
- Desde IntelliJ, en la pestaña del proyecto, haga clic con el botón derecho en el archivo y seleccione `Open In Scene Builder` .
- Desde Eclipse, en la pestaña del proyecto, haga clic con el botón derecho en el archivo y seleccione `Open with Scene Builder` .



Si Scene Builder está correctamente instalado y su ruta se agrega al IDE (ver Comentarios a continuación), abrirá el archivo:



<https://riptutorial.com/es/javafx/topic/5445/constructor-de-escena>

Capítulo 6: CSS

Sintaxis

- `NodeClass` / * selector por clase de nodo * /
- `.someclass` / * selector por clase * /
- `#somedId` / * selector por id * /
- `[selector1]>[selector2]` / * selector para un hijo directo de un nodo que coincida con `selector1` que coincida con `selector2` * /
- `[selector1][selector2]` / * selector para un descendiente de un nodo que coincida con `selector1` que coincida con `selector2` * /

Examples

Usando CSS para el estilo

CSS se puede aplicar en varios lugares:

- en línea (`Node.setStyle`)
- en una hoja de estilo
 - a una `Scene`
 - como hoja de estilo del agente de usuario (no se muestra aquí)
 - como hoja de estilo "normal" para la `Scene`
 - a un `Node`

Esto permite cambiar las propiedades de estilo de los `Nodes` . El siguiente ejemplo demuestra esto:

Clase de aplicación

```
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;
}

```

Rectángulo extensible que agrega nuevas propiedades stylable

JavaFX 8

El siguiente ejemplo muestra cómo agregar propiedades personalizadas que pueden ser estilizadas desde CSS a un `Node` personalizado.

Aquí se `DoubleProperty` 2 `DoubleProperty` s a la clase `Rectangle` para permitir establecer el `width` y `height` desde CSS.

El siguiente CSS podría usarse para diseñar el nodo personalizado:

```

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

```

Nodo personalizado

```

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;  
}  
  
}
```

Lea CSS en línea: <https://riptutorial.com/es/javafx/topic/1581/css>

Capítulo 7: Diálogos

Observaciones

Se agregaron diálogos en la actualización 40 de JavaFX 8.

Examples

TextInputDialog

`TextInputDialog` permite al usuario solicitar que ingrese una sola `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` permite al usuario elegir un elemento de una lista de opciones.

```
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);
```

Alerta

`Alert` es una ventana emergente simple que muestra un conjunto de botones y obtiene un resultado según el botón que el usuario hizo clic:

Ejemplo

Esto permite al usuario decidir si realmente quiere cerrar la etapa 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();
}
```

Tenga en cuenta que el texto del botón se ajusta automáticamente en función de la `Locale`.

Texto de botón personalizado

El texto que se muestra en un botón se puede personalizar creando una instancia de `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();
```

Lea Diálogos en línea: <https://riptutorial.com/es/javafx/topic/3681/dialogos>

Capítulo 8: Diseños

Examples

StackPane

`StackPane` pone a sus hijos en una pila de atrás para adelante.

El orden z de los hijos se define por el orden de la lista de hijos (accesible mediante una llamada a `getChildren()`): el niño número 0 es el último y el último hijo en la parte superior de la pila.

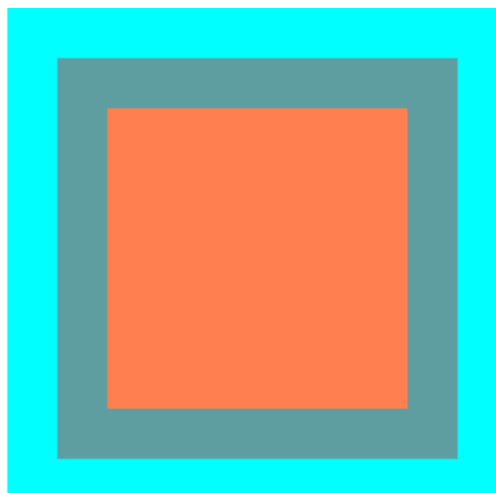
El `stackpane` intenta cambiar el tamaño de cada niño para llenar su propia área de contenido. En el caso de que no se pueda cambiar el tamaño de un niño para llenar el área de `StackPane` (ya sea porque no fue de tamaño variable o su tamaño máximo no lo `StackPane`), entonces se alineará dentro del área usando la propiedad de `alignmentProperty` del `stackpane`, que por defecto es `Pos.CENTER`.

Ejemplo

```
// 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 y VBox

Los diseños de `HBox` y `VBox` son muy similares, ambos presentan a sus hijos en una sola línea.

Características comunes

Si un `HBox` o un `VBox` tienen un borde y / o un conjunto de relleno, entonces el contenido se colocará dentro de esas inserciones.

Distribuyen a cada niño manejado independientemente del valor de propiedad visible del niño; Los niños no administrados son ignorados.

La alineación del contenido está controlada por la propiedad de alineación, que por defecto es `Pos.TOP_LEFT`.

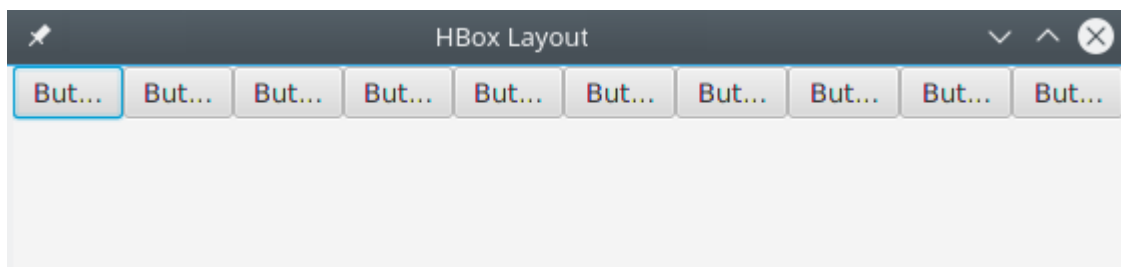
HBox

`HBox` distribuye a sus hijos en una sola fila horizontal de izquierda a derecha.

`HBox` cambiará el tamaño de los niños (si se puede cambiar el tamaño) **a su ancho preferido** y usa su propiedad `fillHeight` para determinar si cambiar el tamaño de sus alturas para llenar su propia altura o mantener sus alturas a sus preferidas (`fillHeight` por defecto es verdadero).

Creando un HBox

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



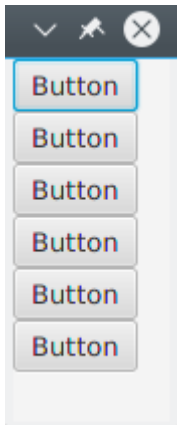
VBox

`VBox` expone a sus hijos en una sola columna vertical de arriba a abajo.

`VBox` cambiará el tamaño de los niños (si se puede cambiar el tamaño) **a sus alturas preferidas** y utiliza su propiedad `fillWidth` para determinar si cambiar el tamaño de sus anchos para completar su propio ancho o mantener sus anchos a su preferido (el valor predeterminado de `fillWidth` es verdadero).

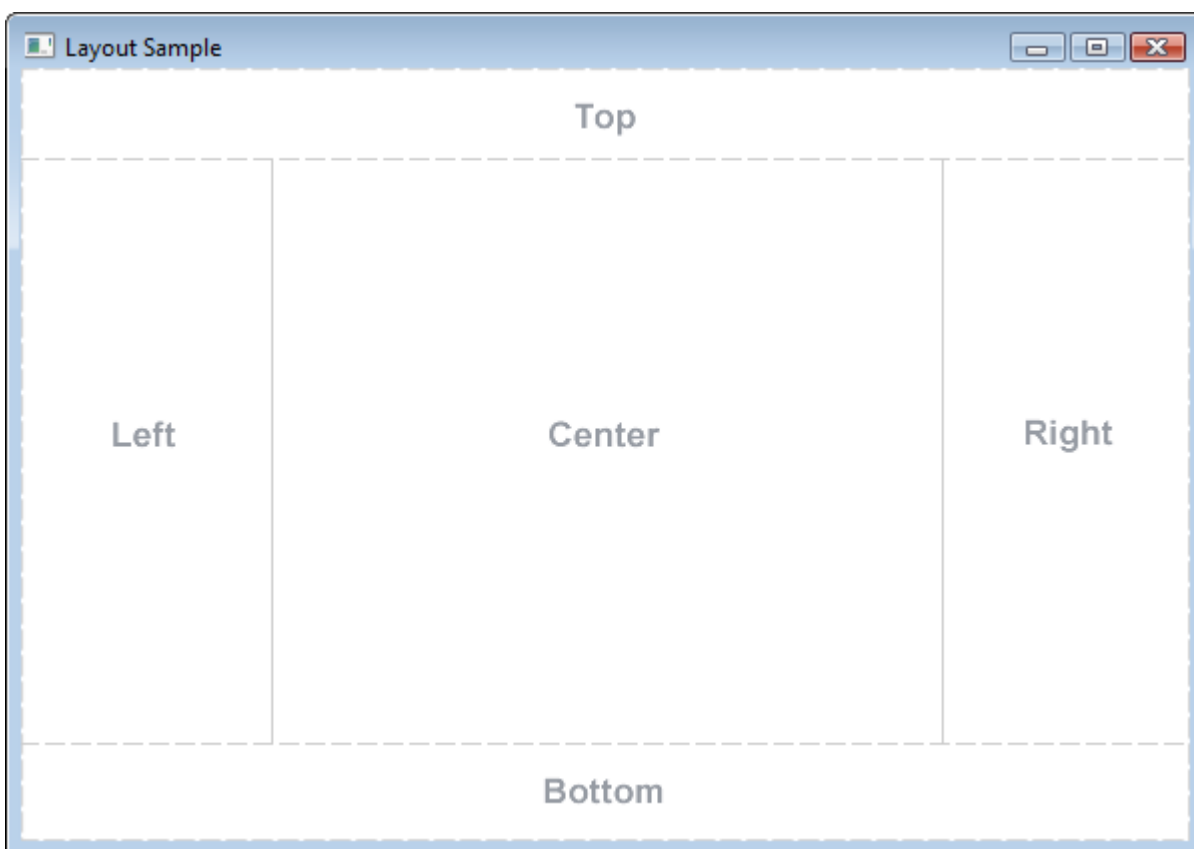
Creando un VBox

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



BorderPane

El `BorderPane` está separado en cinco áreas diferentes.



Las áreas de borde (`Top` , `Right` , `Bottom` , `Left`) tienen un tamaño preferido según su contenido. Por defecto, solo tomarán lo que necesiten, mientras que el área del `Center` tomará el espacio restante. Cuando las áreas fronterizas están vacías, no ocupan espacio.

Cada área puede contener solo un elemento. Se puede agregar utilizando los métodos `setTop(Node)` , `setRight(Node)` , `setBottom(Node)` , `setLeft(Node)` , `setCenter(Node)` . Puede utilizar otros diseños para colocar más de un elemento en un área única.

```
//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

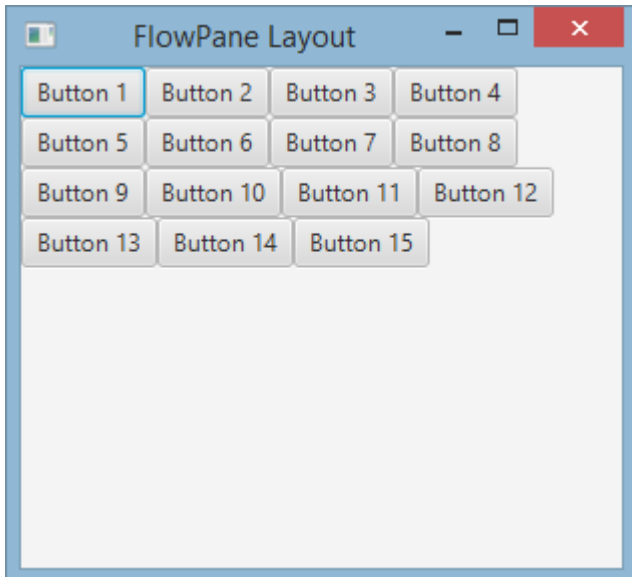
FlowPane establece los nodos en filas o columnas según el espacio disponible horizontal o vertical disponible. Envuelve los nodos a la siguiente línea cuando el espacio horizontal es menor que el total de los anchos de todos los nodos; envuelve los nodos a la siguiente columna cuando el espacio vertical es menor que el total de las alturas de todos los nodos. Este ejemplo ilustra el diseño horizontal predeterminado:

```

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

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

```



`FlowPane` predeterminado de `FlowPane` :

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

`FlowPane` adicionales `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.
```

La adición de nodos al diseño utiliza los métodos `add()` o `addAll()` del `Pane` principal:

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

Por defecto, un `FlowPane` despliega nodos secundarios de izquierda a derecha. Para cambiar la alineación del flujo, llame al método `setAlignment()` pasando un valor enumerado de tipo `Pos`.

Algunas alineaciones de flujo de uso común:

```
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` distribuye a sus hijos dentro de una cuadrícula flexible de filas y columnas.

Hijos del GridPane

Un niño puede colocarse en cualquier lugar dentro de `GridPane` y puede abarcar varias filas / columnas (el intervalo predeterminado es 1) y su ubicación dentro de la cuadrícula se define por sus restricciones de diseño:

Restricción	Descripción
<code>columnIndex</code>	Columna donde comienza el área de diseño del niño.
<code>rowIndex</code>	fila donde comienza el área de diseño del niño.
<code>columnSpan</code>	el número de columnas que abarca el área de diseño del niño horizontalmente.
<code>rowSpan</code>	el número de filas que abarca el área de diseño del niño verticalmente.

No es necesario especificar el número total de filas / columnas por adelantado ya que la cuadrícula expandirá / contraerá automáticamente la cuadrícula para acomodar el contenido.

Añadiendo niños al GridPane

Para agregar nuevos `Node` a un `GridPane` las **restricciones de diseño** en los hijos deben establecerse utilizando el método estático de la clase `GridPane`, luego esos niños pueden agregarse a una instancia de `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` proporciona métodos convenientes para combinar estos pasos:

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

La clase `GridPane` también proporciona métodos de establecimiento estático para establecer la **fila** y la **columna** de elementos secundarios:

```
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
```

Tamaño de columnas y filas

De forma predeterminada, las filas y columnas se ajustarán al tamaño de su contenido. En caso de que sea necesario el **control explícito de los tamaños de fila y columna**, las instancias de `RowConstraints` y `ColumnConstraints` se pueden agregar a `GridPane`. Agregar estas dos restricciones cambiará el tamaño del ejemplo anterior para tener la primera columna de 100 píxeles, la segunda columna de 200 píxeles de longitud.

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

Por defecto, `GridPane` cambiará el tamaño de las filas / columnas a sus tamaños preferidos, incluso si el `gridpane` se redimensiona más que su tamaño preferido. Para admitir **los tamaños dinámicos de columna / fila**, ambas clases de contención proporcionan tres propiedades: tamaño mínimo, tamaño máximo y tamaño preferido.

Además, `ColumnConstraints` proporciona `setHGrow` y `RowConstraints` proporciona los métodos `setVGrow` para **afectar la prioridad del crecimiento y la reducción**. Las tres prioridades predefinidas son:

- **Prioridad . SIEMPRE:** siempre trate de crecer (o reducir), compartiendo el aumento (o disminución) en el espacio con otras áreas de diseño que tengan un aumento (o reducción) de SIEMPRE
- **Prioridad . ALGUNAS VECES :** Si no hay otras áreas de diseño con crecimiento (o reducción) establecido SIEMPRE o si esas áreas de diseño no absorbieron todo el espacio aumentado (o disminuido), compartirán el aumento (o disminución) en el espacio con otra área de diseño de ALGUNAS VECES.
- **Prioridad . NUNCA:** el área de diseño nunca crecerá (o disminuirá) cuando haya un aumento (o disminución) en el espacio disponible en la región.

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

La columna definida anteriormente tiene un tamaño mínimo de 100 píxeles y siempre intentará crecer hasta que alcance su máximo ancho de 300 píxeles.

También es posible definir el **tamaño del porcentaje** para filas y columnas. El siguiente ejemplo define un `GridPane` donde la primera columna llena el 40% del ancho del `gridpane`, la segunda llena el 60%.

```
GridPane gridpane = new GridPane();
ColumnConstraints column1 = new ColumnConstraints();
column1.setPercentWidth(40);
ColumnConstraints column2 = new ColumnConstraints();
```

```
column2.setPercentWidth(60);
gridpane.getColumnConstraints().addAll(column1, column2);
```

Alineación de elementos dentro de las celdas de la cuadrícula.

La alineación de los `Node` se puede definir utilizando el `setHalignment` (horizontal) de `ColumnConstraints` clase `ColumnConstraints` y el `setValignment` (vertical) de la clase `RowConstraints`.

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

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

TilePane

El diseño del panel de mosaico es similar al diseño `FlowPane`. `TilePane` coloca todos los nodos en una cuadrícula en la que cada celda o mosaico tiene el mismo tamaño. Organiza los nodos en filas y columnas ordenadas, ya sea horizontal o 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 b1 = new Button("Buttons");
        root.getChildren().add(b1);
        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);
```

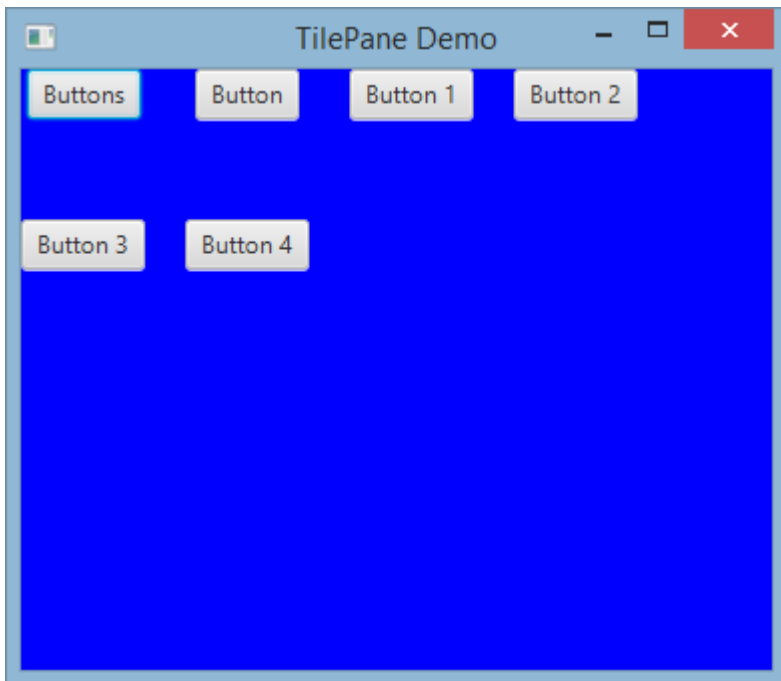
```

        primaryStage.show();
    }

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

```

salida



Para crear Tilepane

```

TilePane root = new TilePane();

```

setHgap () Y el método setVgap () se usa para hacer espacio entre la columna y la columna. También podemos establecer las columnas para el diseño utilizando

```

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

```

AnchorPane

AnchorPane a es un diseño que permite colocar el contenido a una distancia específica de sus lados.

Hay 4 métodos para configurar y 4 métodos para obtener las distancias en **AnchorPane** . El primer parámetro de estos métodos es el **Node** secundario. El segundo parámetro de los configuradores es el valor **Double** a utilizar. Este valor puede ser **null** indica que no hay restricciones para el lado dado.

método de establecimiento	método de obtención
setBottomAnchor	getBottomAnchor
setLeftAnchor	getLeftAnchor
setRightAnchor	getRightAnchor
setTopAnchor	getTopAnchor

En el siguiente ejemplo se colocan nodos a distancias especificadas de los lados.

La región `center` también se redimensiona para mantener las distancias especificadas de los lados. Observe el comportamiento cuando la ventana se redimensiona.

```
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();  
}
```

Lea Diseños en línea: <https://riptutorial.com/es/javafx/topic/2121/disenos>

Capítulo 9: Enhebrado

Examples

Actualizando la interfaz de usuario usando Platform.runLater

Las operaciones de larga ejecución no deben ejecutarse en el subproceso de la aplicación JavaFX, ya que esto evita que JavaFX actualice la interfaz de usuario, lo que resulta en una interfaz de usuario congelada.

Además, cualquier cambio en un `Node` que sea parte de un gráfico de escena "en vivo" **debe** ocurrir en el hilo de la aplicación JavaFX. `Platform.runLater` se puede usar para ejecutar esas actualizaciones en el subproceso de la aplicación JavaFX.

El siguiente ejemplo muestra cómo actualizar un `Node Text` repetidamente desde un hilo diferente:

```
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);
}
}

```

Agrupar las actualizaciones de la interfaz de usuario

El siguiente código hace que la IU no responda por un corto tiempo después de hacer clic en el botón, ya que se usan demasiadas llamadas de `Platform.runLater`. (Intente desplazar el `ListView` inmediatamente después de hacer clic en el botón).

```

@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();
}

```

Para evitar esto en lugar de usar una gran cantidad de actualizaciones, el siguiente código usa un `AnimationTimer` para ejecutar la actualización solo una vez por fotograma:

```

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();
        }
    }
}

```

lo que permite agrupar las actualizaciones usando la clase `Updater` :

```

private final Updater updater = new Updater();

...

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

```

Cómo utilizar el servicio JavaFX

En lugar de ejecutar tareas intensivas en el `JavaFX Thread` que se debe hacer en un `Service` .
¿Entonces qué es básicamente un [Servicio](#) ?

Un servicio es una clase que crea un nuevo `Thread` cada vez que lo está iniciando y le está

pasando una **tarea** para que realice algún trabajo. El servicio puede devolver o no un valor.

A continuación se muestra un ejemplo típico de JavaFX Service que está realizando un trabajo y devuelve un `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;
}
};
}
}
```

Lea Enhebrado en línea: <https://riptutorial.com/es/javafx/topic/2230/enhebrado>

Capítulo 10: Enlaces JavaFX

Examples

Enlace de propiedad simple

JavaFX tiene una API de enlace, que proporciona formas de vincular una propiedad a la otra. Esto significa que cada vez que se cambia el valor de una propiedad, el valor de la propiedad enlazada se actualiza automáticamente. Un ejemplo de enlace simple:

```
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
}
```

También puede vincular una propiedad primitiva aplicando una suma, resta, división, etc.:

```
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'
}
```

Cualquier objeto se puede poner en SimpleObjectProperty:

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

Es posible crear enlaces bidireccionales. En este caso, las propiedades dependen unas de otras.

```
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'
}
```

Lea Enlaces JavaFX en línea: <https://riptutorial.com/es/javafx/topic/7014/enlaces-javafx>

Capítulo 11: FXML y controladores

Sintaxis

- `xmlns:fx = " http://javafx.com/fxml " // declaración de espacio de nombres`

Examples

Ejemplo FXML

Un documento FXML simple que describe un `AnchorPane` contiene un botón y un nodo de etiqueta:

```
<?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>
```

Este archivo FXML de ejemplo está asociado con una clase de controlador. La asociación entre el FXML y la clase del controlador, en este caso, se realiza especificando el nombre de la clase como el valor del atributo `fx:controller` en el elemento raíz del FXML:

`fx:controller="com.example.FXMLDocumentController"` . La clase del controlador permite que el código Java se ejecute en respuesta a las acciones del usuario en los elementos de la IU definidos en el archivo 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
}
}

```

Se puede `FXMLLoader` un `FXMLLoader` para cargar el archivo 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();
    }

}

```

El método de `load` realiza varias acciones, y es útil para entender el orden en que suceden. En este sencillo ejemplo:

1. El `FXMLLoader` lee y analiza el archivo FXML. Crea objetos correspondientes a los elementos definidos en el archivo y toma nota de los atributos `fx:id` definidos en ellos.
2. Dado que el elemento raíz del archivo FXML definió un atributo `fx:controller`, `FXMLLoader` crea una *nueva instancia* de la clase que especifica. De forma predeterminada, esto sucede al invocar el constructor sin argumentos en la clase especificada.
3. Todos los elementos con atributos `fx:id` definidos que tienen campos en el controlador con nombres de campo coincidentes y que son `public` (no recomendados) o anotados en `@FXML` (recomendado) se "inyectan" en esos campos correspondientes. Entonces, en este ejemplo, ya que hay una `Label` en el archivo FXML con `fx:id="label"` y un campo en el controlador definido como

```

@FXML
private Label label ;

```

el campo de `label` se inicializa con la instancia de `Label` creada por `FXMLLoader`.

4. Los controladores de eventos se registran con cualquier elemento en el archivo `onXXX="#..."` con las `onXXX="#..."` definidas. Estos controladores de eventos invocan el método especificado en la clase de controlador. En este ejemplo, dado que el `Button` tiene `onAction="#handleButtonAction"`, y el controlador define un método

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

cuando se dispara una acción en el botón (por ejemplo, el usuario lo presiona), se invoca este método. El método debe tener un tipo de retorno `void` y puede definir un parámetro que coincida con el tipo de evento (`ActionEvent` en este ejemplo) o no puede definir ningún parámetro.

5. Finalmente, si la clase del controlador define un método de `initialize`, se invoca este método. Observe que esto sucede después de que se hayan inyectado los campos `@FXML`, por lo que se puede acceder a ellos de manera segura con este método y se inicializarán con las instancias correspondientes a los elementos en el archivo FXML. El método `initialize()` puede tomar ningún parámetro o puede tomar una `URL` y un `ResourceBundle`. En este último caso, estos parámetros se rellenarán con la `URL` representa la ubicación del archivo FXML y cualquier conjunto de `ResourceBundle` en el `FXMLLoader` través de `loader.setResources(...)`. Cualquiera de estos puede ser `null` si no se establecieron.

Controladores anidados

No es necesario crear la IU completa en un solo FXML utilizando un solo controlador.

La etiqueta `<fx:include>` se puede usar para incluir un archivo fxml en otro. El controlador del fxml incluido se puede inyectar en el controlador del archivo de inclusión al igual que cualquier otro objeto creado por `FXMLLoader`.

Esto se hace agregando el atributo `fx:id` al elemento `<fx:include>`. De esta manera, el controlador del fxml incluido se inyectará en el campo con el nombre `<fx:id value>Controller`.

Ejemplos:

fx: valor de id	nombre de campo para inyección
foo	fooController
respuesta42	answer42Controller
xYz	xYzController

Muestra fxmIs

Mostrador

Este es un fxml que contiene un `StackPane` con un nodo de `Text`. El controlador para este archivo

FXML permite obtener el valor del contador actual, así como incrementar el contador:

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;
    }
}
```

Incluyendo FXML

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

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

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

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

Controlador externo

El controlador del fxml incluido se inyecta en este controlador. Aquí, el controlador para el evento `onAction` para el `Button` se utiliza para incrementar el contador.

```
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();
    }

}
```

Los fxmls se pueden cargar de esta manera, asumiendo que el código se llama desde una clase en el mismo paquete que `outer.fxml` :

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

Definir bloques y

A veces, un elemento debe crearse fuera de la estructura de objeto habitual en el fxml.

Aquí es donde entran en juego los *bloques* definidos:

Los contenidos dentro de un elemento `<fx:define>` no se agregan al objeto creado para el elemento padre.

Cada elemento hijo de `<fx:define>` necesita un atributo `fx:id`.

Los objetos creados de esta manera pueden ser referenciados posteriormente usando el elemento `<fx:reference>` o usando el enlace de expresión.

El elemento `<fx:reference>` se puede usar para hacer referencia a cualquier elemento con un atributo `fx:id` que se maneja antes de que el elemento `<fx:reference>` se maneje utilizando el

mismo valor que el atributo `fx:id` del elemento referenciado en el Atributo `source` del 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>
```

Pasando datos a FXML - accediendo al controlador existente

Problema: algunos datos deben pasarse a una escena cargada desde un fxml.

Solución

Especifique un controlador utilizando el atributo `fx:controller` y obtenga la instancia del controlador creada durante el proceso de carga de la instancia de `FXMLLoader` utilizada para cargar el fxml.

Agregue métodos para pasar los datos a la instancia del controlador y maneje los datos en esos métodos.

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>
```

Controlador

```
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);
    }

}
```

Código utilizado para cargar el 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);
```

Pasando datos a FXML - Especificando la instancia del controlador

Problema: algunos datos deben pasarse a una escena cargada desde un fxml.

Solución

Configure el controlador utilizando la instancia de `FXMLLoader` usa más adelante para cargar el fxml.

Asegúrese de que el controlador contenga los datos relevantes antes de cargar el fxml.

Nota: en este caso, el archivo fxml no debe contener el atributo `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>
```

Controlador

```
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);
    }

}
```

Código utilizado para cargar el 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();
```

Pasando parámetros a FXML - usando un controllerFactory

Problema: algunos datos deben pasarse a una escena cargada desde un fxml.

Solución

Especifique una fábrica de controladores que sea responsable de crear los controladores. Pasar los datos a la instancia del controlador creado por la fábrica.

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>
```

Controlador

```
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);
    }

}
```

Código utilizado para cargar el fxml

Cadena de datos = "¡Hola mundo!";

```
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();

```

Esto puede parecer complejo, pero puede ser útil, si el fxml debería poder decidir, qué clase de controlador necesita.

Creación de instancias en FXML

La siguiente clase se utiliza para demostrar cómo se pueden crear instancias de clases:

JavaFX 8

La anotación en `Person(@NamedArg("name") String name)` debe eliminarse, ya que la anotación `@NamedArg` no está disponible.

```

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();
    }
}

```

Suponga que la clase `Person` ya se ha inicializado antes de cargar el fxml.

Una nota sobre las importaciones.

En el siguiente ejemplo de fxml, la sección de importaciones quedará fuera. Sin embargo, el fxml debe comenzar con

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

seguido de una sección de importaciones que importa todas las clases utilizadas en el archivo fxml. Esas importaciones son similares a las importaciones no estáticas, pero se agregan como instrucciones de procesamiento. **Incluso las clases del paquete `java.lang` necesitan ser importadas.**

En este caso se deben agregar las siguientes importaciones:

```

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

```

JavaFX 8

`@NamedArg` **anotado** `@NamedArg`

Si hay un constructor donde todos los parámetros se anotan con `@NamedArg` y todos los valores de las anotaciones de `@NamedArg` están presentes en el fxml, el constructor se usará con esos parámetros.

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

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

Ambos dan como resultado la siguiente salida de consola, si está cargada:

```
Person(String)
```

No args constructor

Si no hay un constructor anotado de `@NamedArg` adecuado, se utilizará el constructor que no toma parámetros.

Elimine la anotación `@NamedArg` del constructor e intente cargar.

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

Esto utilizará el constructor sin parámetros.

Salida:

```
Person()
setter
```

`fx:value` atributo de `fx:value`

El atributo `fx:value` se puede usar para pasar su valor a un método `valueOf static` que toma un parámetro `String` y devuelve la instancia a usar.

Ejemplo

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

Salida:

```
valueOf
Person(String)
```

`fx:factory`

El atributo `fx:factory` permite la creación de objetos utilizando métodos `static` arbitrarios que no toman parámetros.

Ejemplo

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

Salida:

```
createPerson
Person()
setter
```

<fx:copy>

Usando `fx:copy` se puede invocar un constructor de copia. Especificar el `fx:id` de otro El atributo `source` de la etiqueta invocará al constructor de copia con ese objeto como parámetro.

Ejemplo:

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

Salida

```
Person(Person)
getter
```

fx:constant

`fx:constant` permite obtener un valor de un campo `static final`.

Ejemplo

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

no producirá ningún resultado, ya que esto solo hace referencia a `JOHN` que se creó al inicializar la clase.

Configuración de propiedades

Hay varias formas de agregar datos a un objeto en fxml:

etiqueta `<property>`

Una etiqueta con el nombre de una propiedad se puede agregar como elemento secundario de un elemento utilizado para crear una instancia. El elemento secundario de esta etiqueta se asigna a la propiedad mediante el establecedor o se agrega al contenido de la propiedad (propiedades de lista / mapa de solo lectura).

Propiedad por defecto

Una clase se puede anotar con la anotación `@DefaultProperty`. En este caso, los elementos se pueden agregar directamente como elemento secundario sin utilizar un elemento con el nombre de la propiedad.

`property="value"` **atributo** `property="value"`

Las propiedades se pueden asignar utilizando el nombre de la propiedad como nombre de atributo y el valor como valor de atributo. Esto tiene el mismo efecto que agregar el siguiente elemento como elemento secundario de la etiqueta:

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

colocadores estáticos

Las propiedades también se pueden establecer utilizando establecedores `static`. Estos son métodos `static` llamados `setProperty` que toman el elemento como primer parámetro y el valor para establecer como segundo parámetro. Esos métodos se pueden anular en cualquier clase y se pueden usar usando `ContainingClass.property` lugar del nombre de propiedad habitual.

Nota: actualmente parece necesario tener un método getter estático correspondiente (es decir, un método estático denominado `getProperty` toma el elemento como parámetro en la misma clase que el establecedor estático) para que esto funcione a menos que el tipo de valor sea `String`.

Tipo de coerción

El siguiente mecanismo se utiliza para obtener un objeto de la clase correcta durante las asignaciones, por ejemplo, para adaptarse al tipo de parámetro de un método de establecimiento.

Si las clases son asignables, entonces se usa el valor mismo.

De lo contrario, el valor se convierte de la siguiente manera

Tipo de objetivo	valor utilizado (valor fuente <i>s</i>)
Boolean , boolean	Boolean.valueOf(<i>s</i>)
char , Character	<i>s</i> .toString.charAt(0)
otro tipo primitivo o tipo de envoltura	método apropiado para el tipo de destino, en el caso de que <i>s</i> sea un Number , el <code>valueOf(<i>s</i>.toString())</code> para el tipo de contenedor de lo contrario
BigInteger	BigInteger.valueOf(<i>s</i> .longValue()) es <i>s</i> es un Number , new BigInteger(<i>s</i> .toString()) contrario
BigDecimal	BigDecimal.valueOf(<i>s</i> .doubleValue()) es <i>s</i> es un Number , new BigDecimal(<i>s</i> .toString()) contrario
Número	Double.valueOf(<i>s</i> .toString()) si <i>s</i> .toString() contiene a . , Long.valueOf(<i>s</i> .toString()) contrario
Class	Class.forName(<i>s</i> .toString()) invocado usando el contexto ClassLoader del hilo actual sin inicializar la clase
enumerar	El resultado del método <code>valueOf</code> , que además se convierte en una String mayúsculas separadas por _ insertada antes de cada letra en mayúscula, si <i>s</i> es una String que comienza con una letra en minúscula
otro	el valor devuelto por un método <code>valueOf static</code> en <code>targetType</code> , que tiene un parámetro que coincide con el tipo de <i>s</i> o una superclase de ese tipo

Nota: este comportamiento no está bien documentado y podría estar sujeto a cambios.

Ejemplo

```
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;
    }
}

```

Imprimiendo el resultado de cargar los siguientes rendimientos de archivos `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>

```

```
        </Sample>
    </sample>
</map>
</Sample>
```

Lea FXML y controladores en línea: <https://riptutorial.com/es/javafx/topic/1580/fxml-y-controladores>

Capítulo 12: Gráfico

Examples

Gráfico circular

La clase `PieChart` dibuja datos en forma de círculo que se divide en segmentos. Cada porción representa un porcentaje (parte) para un valor particular. Los datos del gráfico circular se envuelven en objetos `PieChart.Data`. Cada objeto `PieChart.Data` tiene dos campos: el nombre del sector circular y su valor correspondiente.

Constructores

Para crear un gráfico circular, necesitamos crear el objeto de la clase `PieChart`. Dos constructores se dan a nuestra disposición. Uno de ellos crea un gráfico vacío que no mostrará nada a menos que los datos se configuren con el método `setData`:

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

Y el segundo requiere que se pase como parámetro una lista `ObservableList` de `PieChart.Data`.

```
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
```

Datos

Los valores de los sectores de sectores no necesariamente tienen que sumar 100, ya que el tamaño del sector se calculará en proporción a la suma de todos los valores.

El orden en que se agregan las entradas de datos a la lista determinará su posición en el gráfico. Por defecto, se colocan en sentido horario, pero este comportamiento puede revertirse

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

Ejemplo

El siguiente ejemplo crea un gráfico circular simple:

```
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);
    }
}

```

Salida:

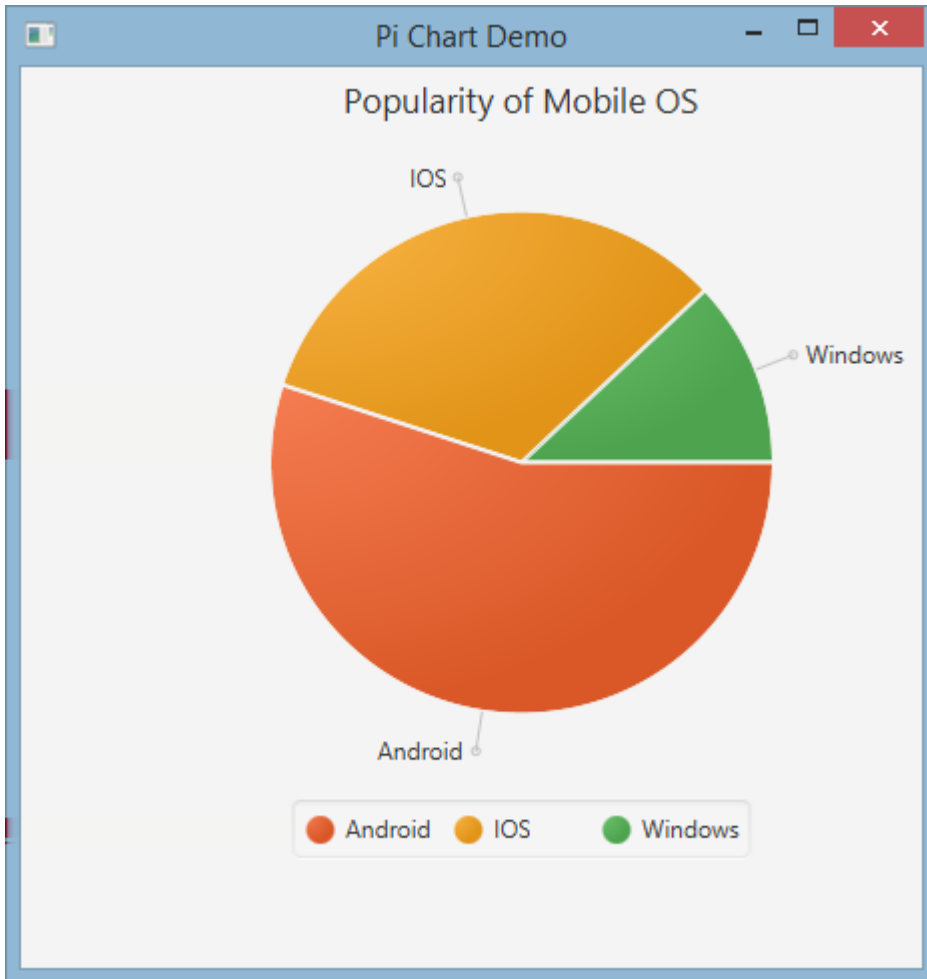


Gráfico circular interactivo

De forma predeterminada, `PieChart` no controla ningún evento, pero este comportamiento se puede cambiar porque cada porción circular es un `Node` JavaFX.

En el siguiente ejemplo, inicializamos los datos, los asignamos al gráfico y luego iteramos sobre el conjunto de datos agregando información sobre herramientas a cada segmento, de modo que los valores, normalmente ocultos, puedan presentarse al usuario.

```
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);  
});
```

Gráfico de línea

La clase `LineChart` presenta los datos como una serie de puntos de datos conectados con líneas rectas. Cada punto de datos se `XYChart.Data` en el objeto `XYChart.Data` , y los puntos de datos se agrupan en `XYChart.Series` .

Cada objeto `XYChart.Data` tiene dos campos, a los que se puede acceder utilizando `getXValue` y `getYValue` , que corresponden a un valor de x y y en un gráfico.

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

Ejes

Antes de crear un `LineChart` necesitamos definir sus ejes. Por ejemplo, el constructor predeterminado, sin argumentos, de una clase `NumberAxis` creará un eje de rango automático que está listo para usar y no requiere configuración adicional.

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

Ejemplo

En el ejemplo completo a continuación, creamos dos series de datos que se mostrarán en el mismo gráfico. Las etiquetas de los ejes, los rangos y los valores de marca están definidos explícitamente.

```
@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);

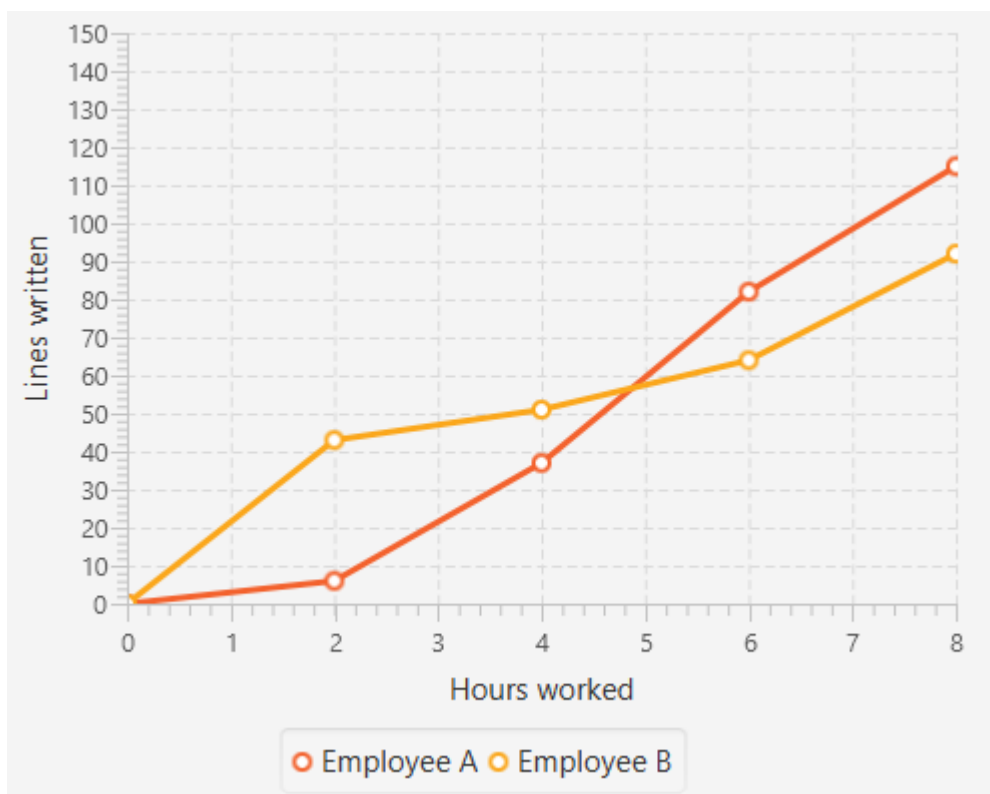
root.getChildren().add(chart);

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

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

```

Salida:



Lea Gráfico en línea: <https://riptutorial.com/es/javafx/topic/2631/grafico>

Capítulo 13: Impresión

Examples

Impresión básica

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

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

Esto se imprime en la impresora predeterminada sin mostrar ningún cuadro de diálogo al usuario. Para usar una impresora que no sea la predeterminada, puede usar

`PrinterJob#createPrinterJob(Printer)` para configurar la impresora actual. Puede usar esto para ver todas las impresoras en su sistema:

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

Impresión con diálogo del sistema

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

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

Lea Impresión en línea: <https://riptutorial.com/es/javafx/topic/5157/impresion>

Capítulo 14: Internacionalización en JavaFX

Examples

Cargando paquete de recursos

JavaFX proporciona una manera fácil de internacionalizar sus interfaces de usuario. Al crear una vista desde un archivo FXML, puede proporcionar al `FXMLLoader` un paquete de recursos:

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

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

Este paquete proporcionado se usa automáticamente para traducir todos los textos en su archivo FXML que comienzan con un `%`. Digamos que el archivo de propiedades `strings_en_UK.properties` contiene la siguiente línea:

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

Si tienes una definición de botón en tu FXML como esta:

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

Recibirá automáticamente la traducción de la clave `ui.button.text`.

Controlador

A Los paquetes de recursos contienen objetos específicos del entorno local. Puede pasar el paquete al `FXMLLoader` durante su creación. El controlador debe implementar la interfaz `Initializable` y anular el método de `initialize(URL location, ResourceBundle resources)`. El segundo parámetro de este método es `ResourceBundle` que se pasa del `FXMLLoader` al controlador y puede ser utilizado por el controlador para traducir textos adicionales o modificar otra información dependiente de la ubicación.

```
public class MyController implements Initializable {

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

Cambio dinámico de idioma cuando la aplicación se está ejecutando

Este ejemplo muestra cómo construir una aplicación JavaFX, donde el idioma se puede cambiar

dinámicamente mientras la aplicación se está ejecutando.

Estos son los archivos de paquete de mensajes utilizados en el ejemplo:

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}
```

La idea básica es tener una clase de utilidad I18N (como alternativa, esto podría implementarse un 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;
    }
}

```

Esta clase tiene una `locale` campo estático que es un objeto de `locale Locale` Java envuelto en una `ObjectProperty` objeto JavaFX, por lo que se pueden crear enlaces para esta propiedad. Los primeros métodos son los métodos estándar para obtener y establecer una propiedad JavaFX.

La `get(final String key, final Object... args)` es el método central que se utiliza para la extracción real de un mensaje de un `ResourceBundle`.

Los dos métodos llamados `createStringBinding` crean un `StringBinding` que está vinculado al campo de `locale` y, por lo tanto, los enlaces cambiarán cada vez que cambie la propiedad de la `locale`. El primero usa sus argumentos para recuperar y formatear un mensaje usando el método de `get` mencionado anteriormente, el segundo se pasa en un valor de `Callable`, que debe producir el nuevo valor de cadena.

Los dos últimos métodos son métodos para crear componentes JavaFX. El primer método se utiliza para crear una `Label` y utiliza un `Callable` para su enlace de cadena interno. El segundo crea un `Button` y utiliza un valor de clave para la recuperación del enlace de cadena.

Por supuesto, se podrían crear muchos más objetos diferentes como `MenuItem` o `ToolTip` pero estos dos deberían ser suficientes para un ejemplo.

Este código muestra cómo se usa esta clase dentro de la aplicación:

```
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);
    }
}

```


La aplicación muestra tres formas diferentes de usar el `StringBinding` creado por la clase `I18N` :

1. el título de la ventana está enlazado directamente mediante un `StringBinding` .
2. Los botones utilizan el método auxiliar con las teclas de mensaje.
3. la etiqueta utiliza el método de ayuda con un `Callable` . Este `Callable` utiliza el método `I18N.get()` para obtener una cadena traducida formateada que contiene el recuento real de conmutadores.

Al hacer clic en un botón, se incrementa el contador y se establece la propiedad de configuración regional de `I18N` , lo que a su vez desencadena el cambio de enlaces de cadenas y, por lo tanto, establece la cadena de la interfaz de usuario en nuevos valores.

Lea Internacionalización en JavaFX en línea:

<https://riptutorial.com/es/javafx/topic/5434/internacionalizacion-en-javafx>

Capítulo 15: Lona

Introducción

Un `Canvas` es un `Node` JavaFX, representado como un área rectangular en blanco, que puede mostrar imágenes, formas y texto. Cada `Canvas` contiene exactamente un objeto `GraphicsContext`, responsable de recibir y almacenar en búfer las llamadas de sorteo, que, al final, se representan en la pantalla mediante `Canvas`.

Examples

Formas básicas

`GraphicsContext` proporciona un conjunto de métodos para dibujar y rellenar formas geométricas. Normalmente, estos métodos necesitan que las coordenadas se pasen como sus parámetros, ya sea directamente o en forma de una matriz de valores `double`. Las coordenadas son siempre relativas al `Canvas`, cuyo origen se encuentra en la esquina superior izquierda.

Nota: `GraphicsContext` no dibujará fuera de los límites del `Canvas`, es decir, intentar dibujar fuera del área del `Canvas` definida por su tamaño y cambiar su tamaño después no producirá ningún resultado.

El siguiente ejemplo muestra cómo dibujar tres formas geométricas rellenas semitransparentes delineadas con un trazo negro.

```
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);
```



Lea Lona en línea: <https://riptutorial.com/es/javafx/topic/8935/lona>

Capítulo 16: Paginación

Examples

Creando una paginación

Las paginaciones en JavaFX utilizan una devolución de llamada para obtener las páginas utilizadas en la animación.

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

Esto crea una lista infinita de botones numerados como 0.. ya que el constructor cero arg crea una paginación infinita. `setPageFactory` toma una devolución de llamada que toma un int, y devuelve el nodo que queremos en ese índice.

Avance automático

```
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();
```

Esto avanza la paginación cada 5 segundos.

Cómo funciona

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

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

`fiveSecondsWonder` es una línea de tiempo que dispara un evento cada vez que termina un ciclo. En este caso el tiempo de ciclo es de 5 segundos.

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

Marque la paginación.

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

Establecer la línea de tiempo para ejecutar para siempre.

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

Crear una paginación de imágenes.

```
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)));
```

Tenga en cuenta que las rutas deben ser direcciones URL, no rutas del sistema de archivos.

Cómo funciona

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

Todo lo demás es solo pelusa, aquí es donde está sucediendo el verdadero trabajo.

`setPageFactory` toma una devolución de llamada que toma un int, y devuelve el nodo que queremos en ese índice. La primera página se asigna al primer elemento de la lista, la segunda al segundo elemento de la lista y así sucesivamente.

Lea Paginación en línea: <https://riptutorial.com/es/javafx/topic/5165/paginacion>

Capítulo 17: Propiedades y observables

Observaciones

Las propiedades son observables y se pueden agregar oyentes. Se utilizan constantemente para las propiedades de `Node` s.

Examples

Tipos de propiedades y nombres

Propiedades estandar

Dependiendo del tipo de propiedad, hay hasta 3 métodos para una sola propiedad. Deje que `<property>` indique el nombre de una propiedad y `<Property>` el nombre de la propiedad con una primera letra en mayúscula. Y sea `T` el tipo de propiedad; para envolturas primitivas usamos el tipo primitivo aquí, por ejemplo, `String` para `StringProperty` y `double` para `ReadOnlyDoubleProperty`.

Nombre del método	Parámetros	Tipo de retorno	Propósito
<code><property>Property</code>	<code>()</code>	La propiedad en sí, por ejemplo. <code>DoubleProperty</code> , <code>ReadOnlyStringProperty</code> , <code>ObjectProperty<VPos></code>	devolver la propiedad en sí para agregar oyentes / vinculante
<code>get<Property></code>	<code>()</code>	<code>T</code>	Devolver el valor envuelto en la propiedad.
<code>set<Property></code>	<code>(T)</code>	<code>void</code>	establecer el valor de la propiedad

Tenga en cuenta que el configurador no existe para las propiedades de solo lectura.

Propiedades de la lista de solo lectura

Las propiedades de la lista de solo lectura son propiedades que proporcionan solo un método de obtención. El tipo de dicha propiedad es `ObservableList`, preferiblemente con un tipo de documento especificado. El valor de esta propiedad nunca cambia; el contenido de la lista `ObservableList` se puede cambiar en su lugar.

Propiedades de mapas de solo lectura

Al igual que en las propiedades de la lista de solo lectura, las propiedades del mapa de solo lectura proporcionan un captador y el contenido puede modificarse en lugar del valor de la propiedad. El getter devuelve un `ObservableMap` .

Ejemplo de `StringProperty`

El siguiente ejemplo muestra la declaración de una propiedad (`StringProperty` en este caso) y muestra cómo agregar un `ChangeListener` a ella.

```
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");
    }
}
```

Ejemplo de `ReadOnlyIntegerProperty`

Este ejemplo muestra cómo usar una propiedad de contenedor de solo lectura para crear una propiedad en la que no se puede escribir. En este caso, el `cost` y el `price` pueden modificarse, pero el `profit` siempre será el `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);
    }

}
```

Lea Propiedades y observables en línea: <https://riptutorial.com/es/javafx/topic/4436/propiedades-y-observables>

Capítulo 18: ScrollPane

Introducción

El ScrollPane es un control que ofrece una vista dinámica de su contenido. Esta visión se controla de varias maneras; (botón de aumento-decremento / rueda del mouse) para tener una vista integral del contenido.

Examples

A) Tamaño del contenido fijo:

El tamaño del contenido será el mismo que el de su contenedor 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) Tamaño del contenido dinámico:

El tamaño del contenido cambiará dependiendo de los elementos agregados que excedan los límites de contenido en ambos ejes (horizontal y vertical) que se pueden ver al moverse a través de 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.
```

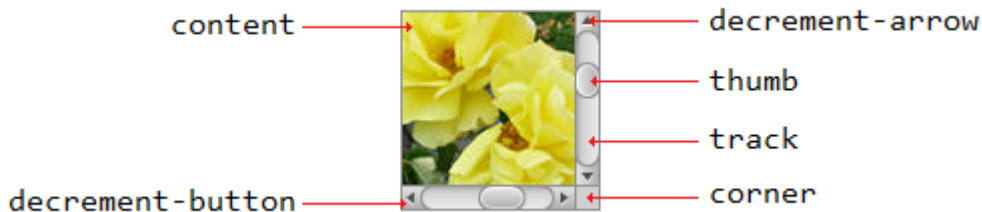
```
scrollpane.setContent(content); // we add the content to the ScrollPane
```

Nota: Aquí no necesitamos ambos métodos (setFitToWidth / setFitToHeight).

Diseñando el ScrollPane:

La apariencia de ScrollPane se puede cambiar fácilmente, teniendo algunas nociones de " CSS " y respetando algunas " *propiedades* " de control y, por supuesto, teniendo algo de " *imaginación* ".

A) Los elementos que componen ScrollPane:



B) propiedades 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{}
```

Lea ScrollPane en línea: <https://riptutorial.com/es/javafx/topic/8259/scrollpane>

Capítulo 19: TableView

Examples

Ejemplo TableView con 2 columnas

Elemento de tabla

La siguiente clase contiene 2 propiedades, un nombre (`String`) y el tamaño (`double`). Ambas propiedades están envueltas en propiedades JavaFX para permitir que `TableView` observe los cambios.

```
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;
    }

}
```

Aplicación de muestra

Esta aplicación muestra un `TableView` con 2 columnas; uno para el nombre y otro para el tamaño de una `Person`. Al seleccionar uno de los `Person`s se agregan los datos a `TextField`s debajo de `TableView` y se permite al usuario editar los datos. Tenga en cuenta que una vez que se comprometa la edición, `TableView` se actualiza automáticamente.

Para cada por cada `TableColumn` añada a la `TableView` un `cellValueFactory` se le asigna. Esta fábrica es responsable de convertir los elementos de la tabla (`Person`) en `ObservableValue`s que contienen el valor que se debe mostrar en la celda de la tabla y que permite que `TableView` escuche los cambios de este valor.

```
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` se puede utilizar como `cellValueFactory` en `TableColumn`. Utiliza la reflexión para acceder a métodos que coinciden con un determinado patrón para recuperar los datos de un elemento de `TableView`:

Ejemplo

```

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

```

El nombre del método que se usa para obtener los datos depende del generador de parámetros para `PropertyValueFactory`.

- **Método de propiedad:** se espera que este tipo de método devuelva un valor `ObservableValue` contenga los datos. Se pueden observar cambios. Deben coincidir con la `<constructor parameter>Property` del patrón `<constructor parameter>Property` y no tomar parámetros.
- **Método de captador:** este tipo de método espera devolver el valor directamente (`String` en el ejemplo anterior). El nombre del método debe coincidir con el patrón `get<Constructor parameter>`. Tenga en cuenta que aquí `<Constructor parameter>` comienza con una *letra mayúscula*. Este método no debería tener parámetros.

Nombres de muestra de métodos.

parámetro constructor (sin comillas)	nombre del metodo de propiedad	nombre del método getter
foo	fooProperty	getFoo
foobar	fooBarProperty	getFooBar
XYZ	XYZProperty	getXYZ
listIndex	listIndexProperty	getListIndex
un valor	aValueProperty	GetValue

Personalizar el aspecto de TableCell dependiendo del artículo

A veces, una columna debe mostrar un contenido diferente al valor de `toString` del elemento de celda. En este caso, las `TableCell` creadas por `cellFactory` de `TableColumn` se personalizan para cambiar el diseño en función del elemento.

Nota importante: `TableView` solo crea las `TableCell` que se muestran en la interfaz de usuario. Los elementos dentro de las celdas pueden cambiar e incluso quedar vacíos. El programador debe tener cuidado de deshacer cualquier cambio que se haya hecho en el `TableCell` cuando se agregó un elemento cuando se eliminó. De lo contrario, es posible que el contenido aún se muestre en una celda donde "no pertenece".

En el ejemplo siguiente, la configuración de un elemento da como resultado el texto que se configura, así como la imagen que se muestra en `ImageView` :

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

Si el elemento se vuelve `null` o la celda se vacía, esos cambios se deshacen al volver a establecer los valores en `null` :

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

El siguiente ejemplo muestra un emoji además de texto en un `TableCell` .

El método `updateItem` se llama cada vez que se cambia el elemento de una `Cell` . La anulación de este método permite reaccionar a los cambios y ajustar el aspecto de la celda. Agregar una escucha a `itemProperty()` de una celda sería una alternativa, pero en muchos casos se extiende `TableCell` .

Tipo de artículo

```
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;
    }
}

```

Código en la clase de aplicación

```

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);
    }
}

```

Clase celular

```

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());
        }
    }
}

```

Añadir botón a TableView

Puede agregar un botón u otro componente javafx a TableView utilizando el

`setCellFactory(Callback value) column` `setCellFactory(Callback value) .`

Aplicación de muestra

En esta aplicación vamos a agregar un botón a TableView. Cuando se hace clic en este botón de columna, se seleccionan los datos en la misma fila que el botón y se imprime su información.

En el método `cellFactory addToTable()`, la `cellFactory` llamada de `cellFactory` es responsable de agregar un botón a la columna relacionada. Definimos el invocador `cellFactory` e implementamos su método de anulación de `call(...)` para obtener `TableCell` con el botón y luego este conjunto de `cellFactory` a la columna relacionada `setCellFactory(..)`. En nuestro ejemplo, esto es `colBtn.setCellFactory(cellFactory)`. SSCCE está abajo:

```
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);

        addToTable();
    }
}
```

```

        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;
}

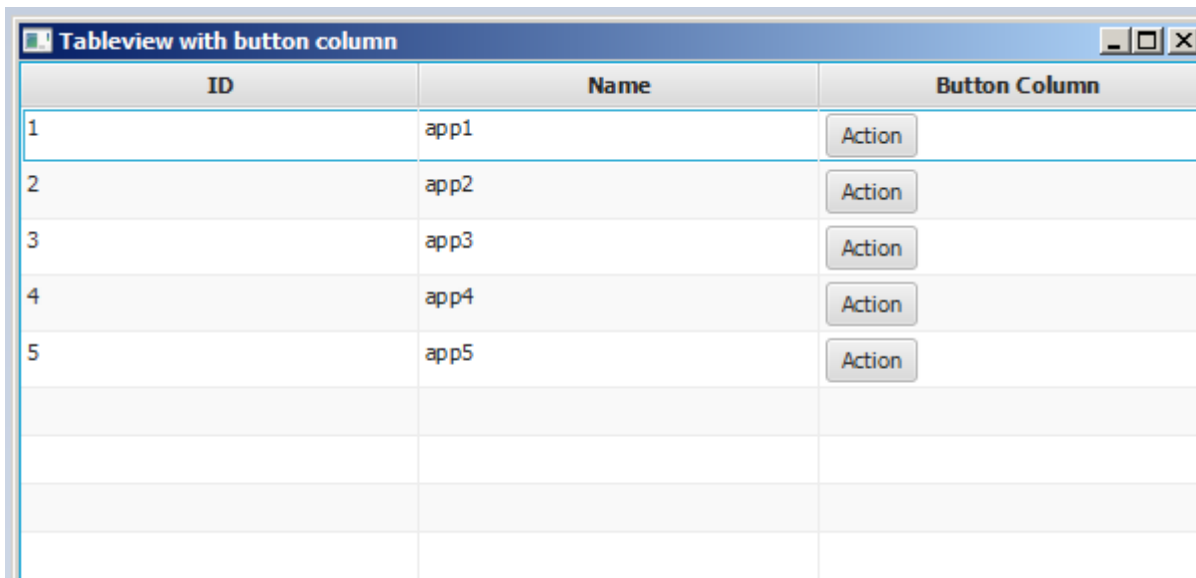
public String getName() {
    return name;
}

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

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

```

Captura de pantalla:



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

Lea TableView en línea: <https://riptutorial.com/es/javafx/topic/2229/tableview>

Capítulo 20: WebView y WebEngine

Observaciones

`WebView` es el nodo JavaFX que está integrado en el árbol de componentes de JavaFX. Administra un `WebEngine` y muestra su contenido.

El `WebEngine` es el motor de navegador subyacente, que básicamente hace todo el trabajo.

Examples

Cargando una pagina

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

`WebView` es el shell de la interfaz de usuario alrededor del `WebEngine`. Casi todos los controles para la interacción sin interfaz de usuario con una página se realizan a través de la clase `WebEngine`.

Obtener el historial de la página de un WebView

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

El historial es básicamente una lista de entradas. Cada entrada representa una página visitada y proporciona acceso a información relevante de la página, como la URL, el título y la fecha en que se visitó la página por última vez.

La lista se puede obtener utilizando el método `getEntries()`. El historial y la lista correspondiente de entradas cambian a medida que `WebEngine` navega por la web. La lista puede expandirse o reducirse dependiendo de las acciones del navegador. Estos cambios pueden ser escuchados por la API `ObservableList` que la lista expone.

El índice de la entrada del historial asociado con la página visitada actualmente se representa mediante `currentIndexProperty()`. El índice actual se puede usar para navegar a cualquier entrada en el historial usando el método `go(int)`. `maxSizeProperty()` establece el tamaño máximo del historial, que es el tamaño de la lista del historial.

A continuación se muestra un ejemplo de cómo [obtener y procesar la Lista de elementos del historial web](#).

Se utiliza un `ComboBox` (`comboBox`) para almacenar los elementos del historial. Al usar un `ListChangeListener` en el `WebHistory` el `ComboBox` se actualiza al `WebHistory` actual. En el `ComboBox` hay un `EventHandler` que redirige a la página seleccionada.

```

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();
    }
});

```

envíe alertas de Javascript desde la página web mostrada al registro de aplicaciones 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()));

```

Comunicación entre la aplicación Java y Javascript en la página web.

Cuando use un WebView para mostrar su propia página web personalizada y esta página contiene Javascript, puede ser necesario establecer una comunicación bidireccional entre el programa Java y el Javascript en la página web.

Este ejemplo muestra cómo configurar dicha comunicación.

La página web mostrará un campo de entrada y un botón. Al hacer clic en el botón, el valor del campo de entrada se envía a la aplicación Java, que lo procesa. Después de procesar un resultado, se envía a Javascript, que a su vez muestra el resultado en la página web.

El principio básico es que para la comunicación de Javascript a Java se crea un objeto en Java que se configura en la página web. Y para la otra dirección, un objeto se crea en Javascript y se extrae de la página web.

El siguiente código muestra la parte de Java, lo guardé todo en un archivo:

```
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());
            }
        }
    }
}

```

Cuando la página se ha cargado, un objeto `JavaConnector` (definido por la clase interna y creado como un campo) se establece en la página web mediante estas llamadas:

```

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

```

El objeto `javascriptConnector` se recupera de la página web con

```

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

```

Cuando se `toLowerCase(String)` método `toLowerCase(String)` del `JavaConnector`, el valor pasado se convierte y luego se devuelve a través del objeto `javascriptConnector`.

Y este es el código html y javascript:

```

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

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

    </main>

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

      var jsConnector = {
        showResult: function (result) {

```

```

        document.getElementById('result').innerHTML = result;
    }
};

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

```

La función `sendToJava` llama al método del `JavaConnector` que fue establecido por el código Java:

```

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

```

y la función llamada por el código Java para recuperar el `javascriptConnector` simplemente devuelve el objeto `jsConnector` :

```

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

function getJsConnector() {
    return jsConnector;
};

```

El tipo de argumento de las llamadas entre Java y Javascript no se limita a cadenas. Más información sobre los posibles tipos y conversión se encuentra en el [documento de API JSObject](#) .

Lea `WebView` y `WebEngine` en línea: <https://riptutorial.com/es/javafx/topic/5156/webview-y-webengine>

Capítulo 21: Windows

Examples

Creando una nueva ventana

Para mostrar algo de contenido en una nueva ventana, se necesita crear un `Stage`. Después de la creación e inicialización `show` debe `showAndWait` a `showAndWait` en el objeto `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: este código debe ejecutarse en el subprocesso de la aplicación JavaFX.

Creación de un diálogo personalizado

Puede crear diálogos personalizados que contienen muchos componentes y realizar muchas funciones en ellos. Se comporta como segunda etapa en el escenario propietario.

En el siguiente ejemplo, una aplicación que muestra a la persona en la vista de tabla del escenario principal y crea una persona en un diálogo (`AddingPersonDialog`) preparada. GUI creadas por `SceneBuilder`, pero pueden crearse mediante códigos Java puros.

Aplicación de muestra:

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>

```

Persona.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();
}

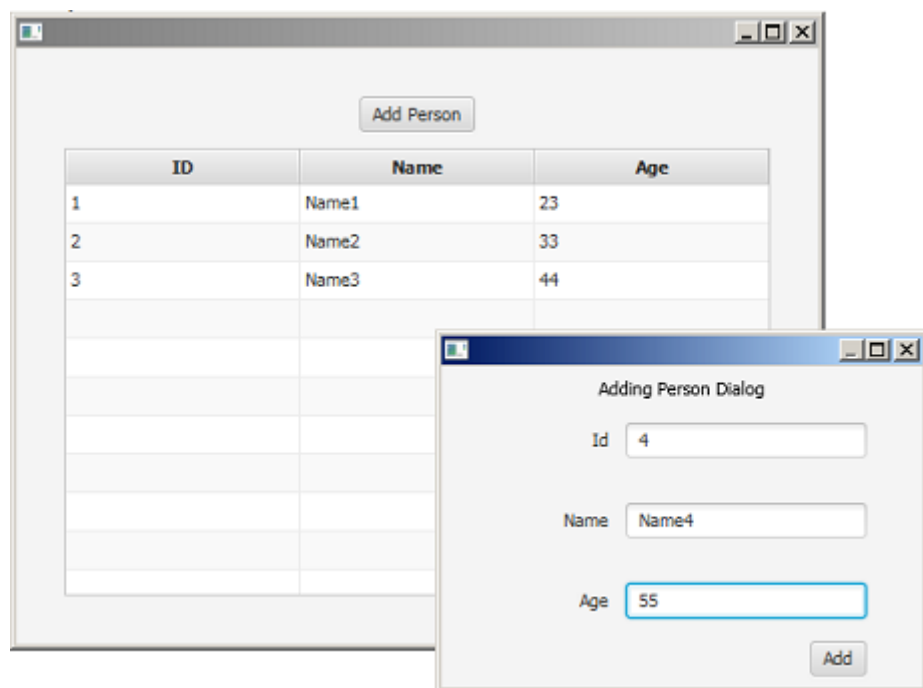
public void setName(String nme) {
    this.name.set(nme);
}

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();
}
}

```



Captura de pantalla

Creación de un diálogo personalizado

Puede crear diálogos personalizados que contienen muchos componentes y realizar muchas funciones en ellos. Se comporta como segunda etapa en el escenario propietario.

En el siguiente ejemplo, una aplicación que muestra a la persona en la vista de tabla del escenario principal y crea una persona en un diálogo (AddingPersonDialog) preparada. GUI

creadas por SceneBuilder, pero pueden crearse mediante códigos Java puros.

Aplicación de muestra:

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>

```

Persona.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();
    }

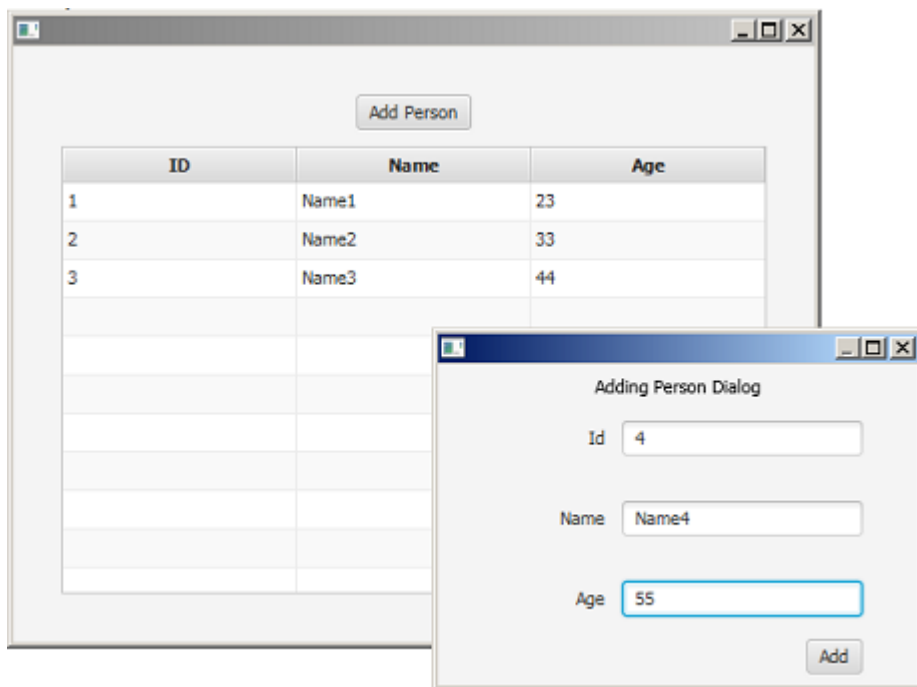
    public void setAge(int age) {
        this.age.set(age);
    }

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

}

```

Captura de pantalla



Lea Windows en línea: <https://riptutorial.com/es/javafx/topic/1496/windows>

Creditos

S. No	Capítulos	Contributors
1	Empezando con javafx	Community , CraftedCart , D3181 , DVarga , fabian , Ganesh , Hendrik Ebbers , Petter Friberg
2	Animación	fabian , J Atkin
3	Botón	Dth , DVarga , J Atkin , Maverick283 , Nico T , Squidward
4	Boton de radio	Nico T
5	Constructor de escena	Ashlyn Campbell , José Pereda
6	CSS	fabian
7	Diálogos	fabian , GltknBtn , Modus Tollens
8	Diseños	DVarga , fabian , Filip Smola , Jinu P C , Sohan Chowdhury , trashgod
9	Enhebrado	Brendan , fabian , GOXR3PLUS , James_D , Koko Essam , sazzy4o
10	Enlaces JavaFX	Alexiy
11	FXML y controladores	D3181 , fabian , James_D
12	Gráfico	Dth , James_D , Jinu P C
13	Impresión	J Atkin , Squidward
14	Internacionalización en JavaFX	ItachiUchiha , Joffrey , Nico T , P.J.Meisch
15	Lona	Dth
16	Paginación	fabian , J Atkin
17	Propiedades y observables	fabian
18	ScrollPane	Bo Halim
19	TableView	Bo Halim , fabian , GltknBtn

20	WebView y WebEngine	fabian , J Atkin , James_D , P.J.Meisch , Squidward
21	Windows	fabian , GltknBtn