



***Técnico en***  
**< DESARROLLO DE SOFTWARE >**

***Programación Orientada a  
Objetos II***

(CC BY-NC-ND 4.0)  
International

Attribution-NonCommercial-NoDerivatives 4.0



## **Atribución**

Usted debe reconocer el crédito de una obra de manera adecuada, proporcionar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que tiene el apoyo del licenciante o lo recibe por el uso que hace.



## **No Comercial**

Usted no puede hacer uso del material con fines comerciales.



## **Sin obra derivada**

Si usted mezcla, transforma o crea un nuevo material a partir de esta obra, no puede distribuir el material modificado.

No hay restricciones adicionales - Usted no puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otros hacer cualquier uso permitido por la licencia.

<http://creativecommons.org/licenses/by-nc-nd/4.0/>



# ***Programación Orientada a Objetos II***

## ***Unidad I***

### **1. Interfaz gráfica de usuario**

Conocida también como GUI por sus siglas en inglés (*Graphical User Interface*). Proporciona al usuario una manera de interactuar con un programa informático haciendo uso de elementos gráficos que representan la información y las acciones que el usuario puede realizar a través de los periféricos de la computadora. A través del tiempo las computadoras adoptaron las GUI como el modo primario de interacción debido a que son más intuitivas para el usuario que las interfaces de texto.

### **2. JavaFX**

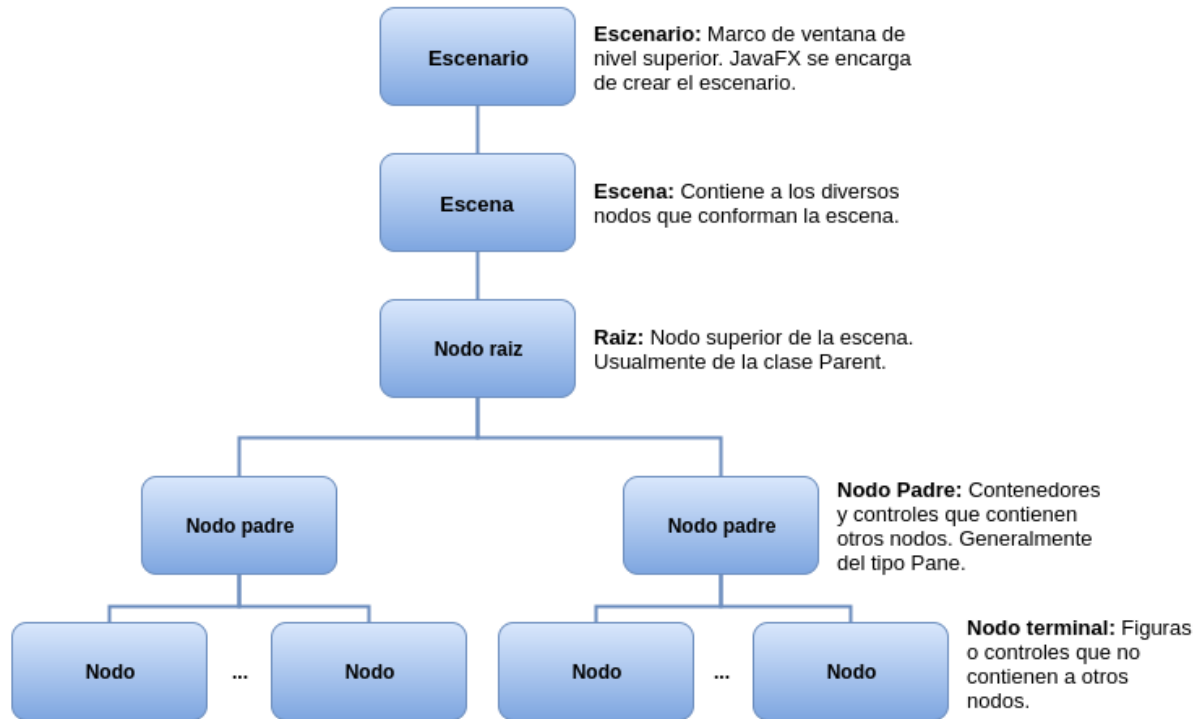
JavaFX inició como un lenguaje de programación declarativo construido a partir de Java. Pero esto implicaba que los desarrolladores después de aprender Java tendrían que aprender un nuevo lenguaje. Después se lanzó JavaFX 2.0 el cual estaba basado en Java APIs, lo que permitió utilizar los mismos comandos de Java. A partir de Java 7, Java FX fue incluido en la versión estándar. Sin embargo con el lanzamiento de Java 11, JavaFX se separó nuevamente del JDK estándar y ahora es parte de la iniciativa *Open Source* de Oracle como OpenJFX.

JavaFX también provee FXML para describir los componentes de las interfaces de usuario. FXML es un lenguaje de marcado declarativo. Las interfaces se pueden escribir directamente o se puede utilizar la herramienta Scene Builder que permite generar FXML con una interfaz gráfica. Con Scene Builder se construyen las interfaces arrastrando y soltando los componentes en un editor visual. Esto permite construir interfaces complicadas con mayor facilidad.

Además JavaFX permite aplicar estilos CSS (hoja de estilos en cascada) a las interfaces gráficas para personalizar los componentes que las conforman.

## Ventanas en JavaFX

Para manejar las ventanas JavaFX utiliza una analogía de un escenario (stage) y una escena (scene). El escenario representa el contenedor de nivel superior para todos los objetos. La escena define una estructura jerárquica de nodos que contiene todos los elementos de una escena. Los nodos son todos los objetos gráficos que se pueden representar en JavaFX por ejemplo: botones, etiquetas, figuras, contenedores de diseños entre otros. A su vez un nodo contenedor de diseño (pane) puede contener otros nodos, lo que permite agrupar varios nodos. El nodo raíz de la aplicación contiene todos los nodos que conforman la escena.



Como primer punto vamos a ver cómo se crea una aplicación con una sola ventana, luego vamos a tratar aplicaciones con múltiples ventanas.

### 3. NetBeans IDE

NetBeans es un Entorno de Desarrollo Integrado libre, usado principalmente para el desarrollo en Java, existe una gran cantidad de módulos que nos permiten expandir su funcionalidad, en particular nos interesa que nos permite trabajar con JavaFX

#### ¿Qué es un IDE?

Un IDE, por sus siglas en inglés, Entorno de Desarrollo Integrado, es un sistema de software que combina muchas herramientas útiles para el desarrollo de software, tales como:

- Editor de código fuente, el que nos permite crear y editar programas. La mayoría de los IDE soportan múltiples lenguajes de programación y el editor de código fuente trae la capacidad de verificar la sintaxis de los comandos.
- Compilación Local, los IDEs también incluyen una forma de compilar y ejecutar los programas que estamos desarrollando, puede ser que tengan incluida una terminal, o incluso pueden llegar a tener emuladores de dispositivos.
- Debugger o Depurador, el *Debugger* es una opción que nos permite correr el programa y detenerlo en determinados sitios, para poder verificar el funcionamiento específico y poder buscar errores, también nos permite ver los recursos y el estado de memoria de un programa.

Los IDEs no son editores de texto, son programas especializados para el desarrollo de software.

## SceneBuilder

Es una herramienta de desarrollo gráfico para JavaFX que nos permite crear interfaces con solo arrastrar y soltar componentes. Se puede integrar con NetBeans en el desarrollo de aplicaciones.

## Maven

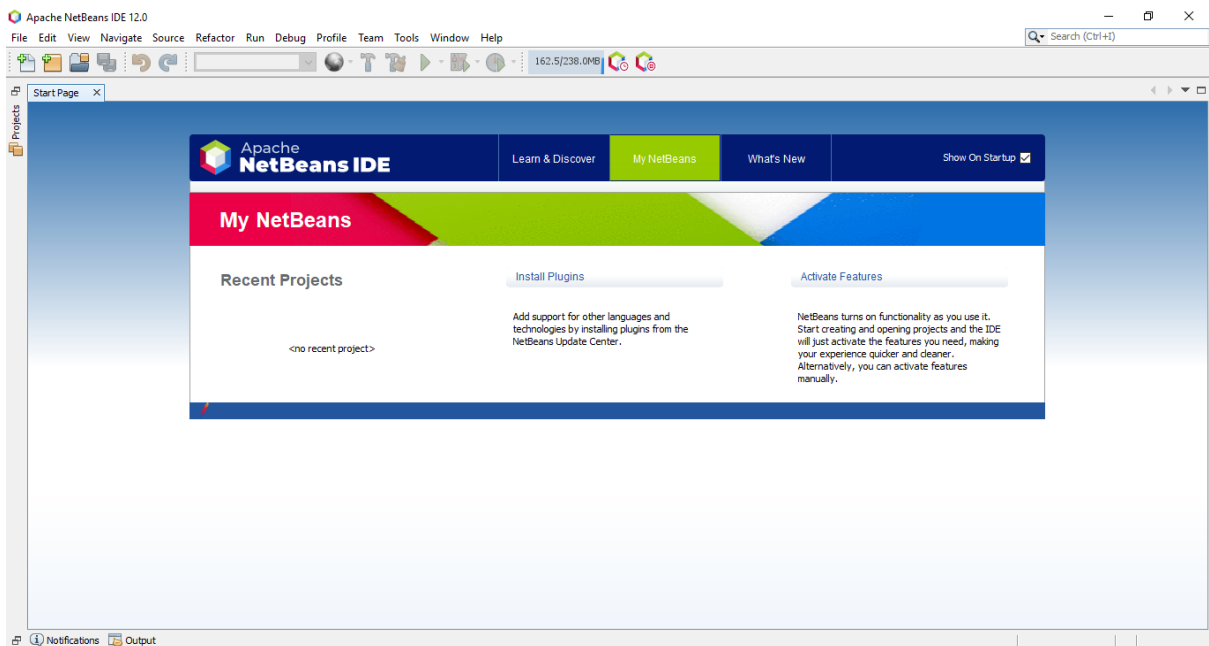
Es una herramienta de gestión de proyectos, maneja los parámetros y librerías que son necesarios para la construcción y ejecución de los proyectos de manera automática, es capaz de descargar y gestionar las dependencias necesarias para el proyecto. Lo usaremos por la facilidad que nos proporciona para agregar librerías a nuestros proyectos.

Para hacer esto se define un archivo que contiene las configuraciones y dependencias del proyecto, el archivo es el llamado POM (Project Object Model) o Modelo de Objetos del Proyecto el cual tiene un formato de XML.

NetBeans trae integrado Maven entonces no hay necesidad de instalarlo.

## Primer vistazo a los componentes

### NetBeans IDE



La primera vez que se inicie el IDE no tendrá gran cosa, nos recibirá una pantalla en la que podremos ver los proyectos recientes

La barra de herramientas es donde podremos construir y correr nuestros proyectos, está dividida en 4 secciones



1. Opciones del proyecto:

- a. **Nuevo Archivo:** Agrega un nuevo archivo al proyecto abierto
- b. **Nuevo Proyecto:** Crea un nuevo proyecto
- c. **Abrir Proyecto:** Abre un proyecto existente
- d. **Guardar Todo:** Guarda todos los cambios en los archivos abiertos

## 2. Opciones de Edición

- a. **Deshacer:** Revierte el último cambio realizado
- b. **Rehacer:** Rehace el cambio eliminado por la opción de deshacer

## 3. Opciones de Ejecución

- a. **Configuración Seleccionada:** Nos permite elegir un perfil de configuración de ejecución, en el cual podemos definir configuraciones como: parámetros de entrada y opciones de ejecución
- b. **Opciones de Navegador,** elegir el navegador en el cual se va a ejecutar el programa, usado en proyectos de JavaScript y Node.js
- c. **Construir:** Compila el proyecto
- d. **Limpiar y Construir:** Elimina las configuraciones guardadas y compila nuevamente
- e. **Ejecutar:** Inicia la ejecución el proyecto
- f. **Iniciar Debugger:** Inicia el proyecto con un *debugger* asociado, lo que nos permite controlar ejecución paso a paso, revisar el valor de las variables, entre otras opciones
- g. **Iniciar Profiler del Proyecto:** Nos permite revisar el estado del uso de la memoria del proyecto actual

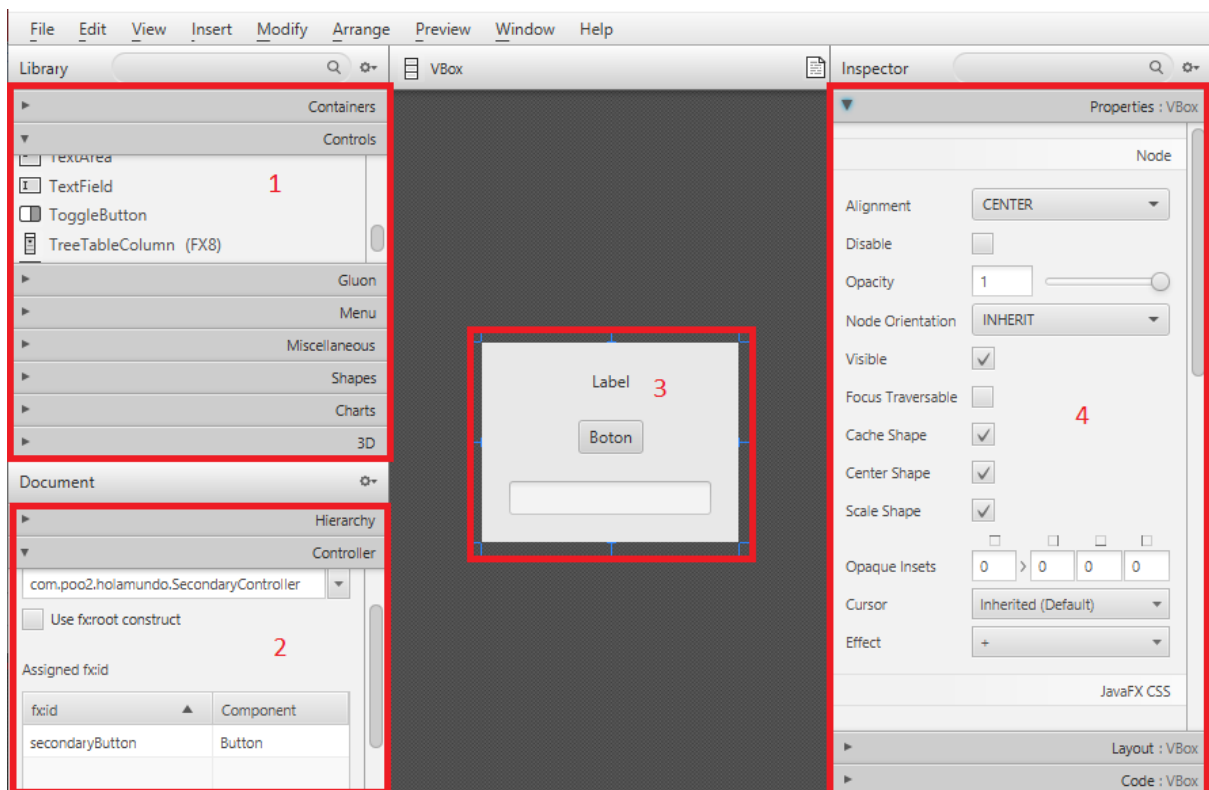
## 4. Opciones de Rendimiento



- a. **Memoria Utilizada:** Nos muestra la memoria que está siendo utilizada por el IDE y los proyectos.
- b. **Iniciar Profiler del IDE:** Nos muestra el estado del uso de la memoria del IDE
- c. **Iniciar / Pausar revisiones de I/O:** Nos permite sincronizar los cambios externos realizados al proyecto con el IDE, por ejemplo si agregamos una imagen nueva a la carpeta del proyecto, esta opción nos la muestra automáticamente en el IDE.

## SceneBuilder

Es un entorno gráfico que nos permite diseñar las ventanas de la interfaz de las aplicaciones



1. **Menú de Componentes:** es la galería de componentes que podemos colocar en nuestras interfaces, incluyen: Campos de Texto, Botones, figuras como rectángulos o círculos, Gráficos de Barras y Gráficos de Pie.
2. **Menú del documento:** Nos muestra opciones relacionadas a los archivos, la opción de jerarquía nos muestra los componentes que podemos encontrar en el archivo actual, mientras que la opción de controlador nos muestra información relacionada al archivo que maneja los eventos de este archivo.
3. **Lienzo:** Es el área donde podemos colocar nuestros componentes para crear la interfaz, seleccionamos un componente del menú de componentes y lo arrastramos hasta el lienzo para colocarlo en nuestra interfaz
4. **El inspector:** Nos muestra opciones relacionadas al componente que tenemos seleccionado, tiene 3 menús que nos muestra diferente información: el primero Propiedades, nos muestra propiedades relacionadas al componente, como alineación, opacidad, estilos de CSS, opciones de accesibilidad, etc. La opción de Layout nos muestra opciones de posicionamiento, tamaño, márgenes, rotación, entre otros. Finalmente la opción de código nos muestra las opciones relacionadas al código, como id del componente, o métodos a ejecutar en distintos escenarios, como click de un botón.

## Maven

Es una herramienta de gestión de librerías y dependencias, esto significa que descarga, compila y empaca programas de Java con las librerías y dependencias de manera automática. Para eso define las configuraciones en un archivo llamado POM.

### ¿Qué es el POM?

Un archivo POM describe la estructura de un proyecto, el POM es un archivo de formato XML, similar a los archivos FXML que usa JavaFX para diseñar las interfaces. Los archivos están compuestos por etiquetas que nos proporcionan información sobre el proyecto.

El POM está compuesto por distintas partes, pero empecemos por el POM básico

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.poo2</groupId>
  <artifactId>mavenproject</artifactId>
  <version>1.0-SNAPSHOT</version>
</project>
```

La primera etiqueta que podemos encontrar es la de project, esta contiene a todas las demás etiquetas, en la etiqueta project podemos encontrar varias declaraciones de xmlns, estas significan XML NameSpace, siempre es seguido por una dirección. La dirección nos indica el conjunto de elementos a los que pertenecen los objetos que declaramos en la estructura del proyecto. Por ejemplo en el POM podemos encontrar mientras que en un archivo FXML podemos encontrar

```
xmlns="http://javafx.com/javafx/11.0.1"
```

Las etiquetas que podemos encontrar dentro de project son

- **GroupId** es el nombre de la organización, grupo o más bien el directorio donde se encuentra el proyecto en cuestión, normalmente sigue las convenciones de nombramiento de paquetes. Hablaremos de ellas más adelante.
- **ArtifactId**: Es el nombre del proyecto

- **Versión:** el número de versión actual del proyecto, cuando agregamos funcionalidades al proyecto que cambian su uso, también debería de cambiar el número de versión del POM
- **ModelVersion:** se refiere a la versión del POM que utiliza, actualmente la versión 4.0.0 es la única que se soporta

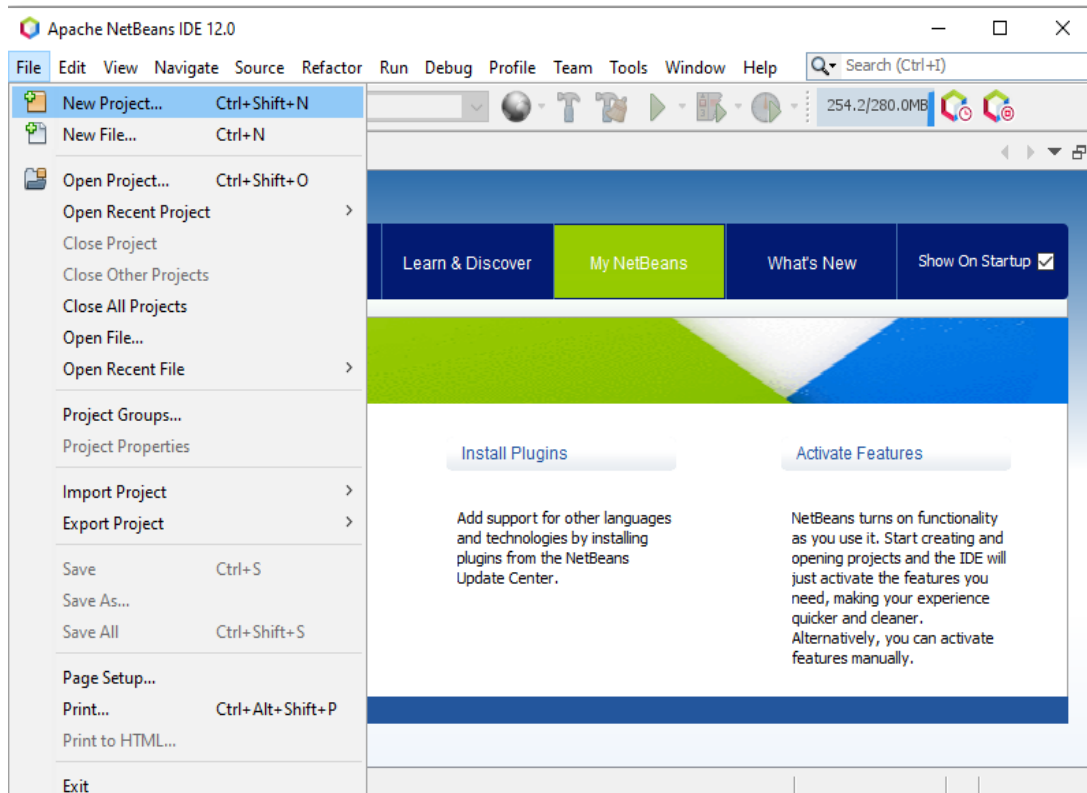
En el POM podemos definir otros módulos, algunas que podemos encontrar son

- **Dependencias:** es el modulo que más usaremos, es donde agregamos librerías externas a nuestro proyecto.
- **build:** son opciones de configuración para la construcción del proyecto, en el podemos encontrar recursos como archivos de configuración o plugins que estemos usando en el proyecto
- **licenses:** La licencia de distribución bajo la cual se maneja el uso del proyecto.

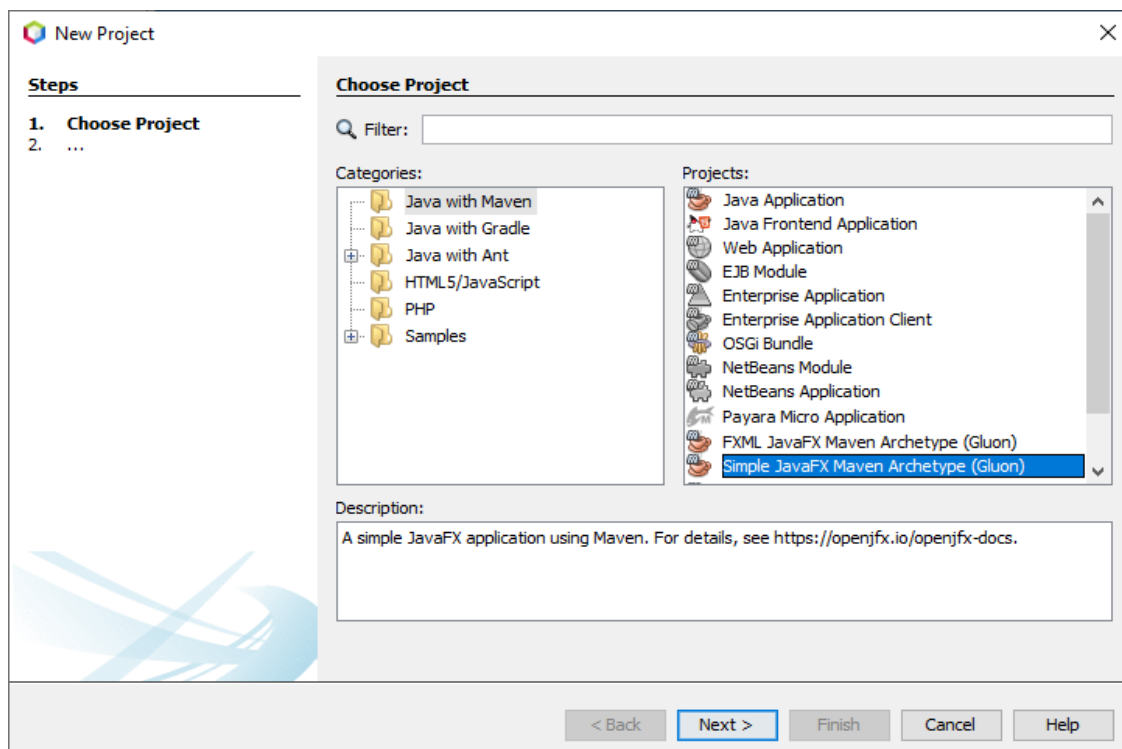
## 4. Creando un proyecto

Vamos a crear un proyecto simple en NetBeans. Recordemos que como prerrequisito se necesita tener Java instalado.

En el IDE, elegimos *Archivo > Nuevo Proyecto*



Como vamos a aprender a usar Maven seleccionamos nuevo proyecto con Maven y luego Proyecto Simple



Veremos la siguiente ventana

New Simple JavaFX Maven Archetype (Gluon)

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name:

Project Location:

Project Folder:

Artifact Id:

Group Id:

Version:

Package:  (Optional)

Additional Creation Properties:

Key	Value
javafx-version	13
javafx-maven-plugin-version	0.0.4
add-debug-configuration	Y

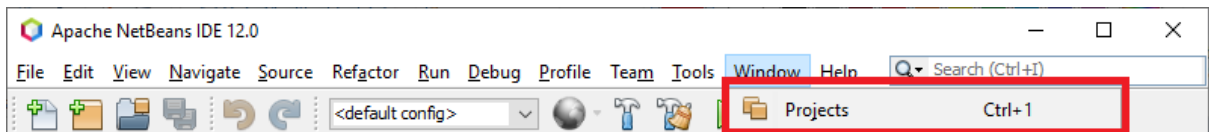
< Back   Next >   **Finish**   Cancel   Help

Aquí solo tenemos que cambiar Project Name, le colocaremos HolaMundo, y GroupId, aquí podemos colocar com.poo2 o puede colocar otro nombre que prefiera. Luego seleccionamos *Finish*

Cuando el proyecto termine de cargar, podemos darle a ejecutar en la barra de herramientas. Nos debería de mostrar una ventana parecida a esta:

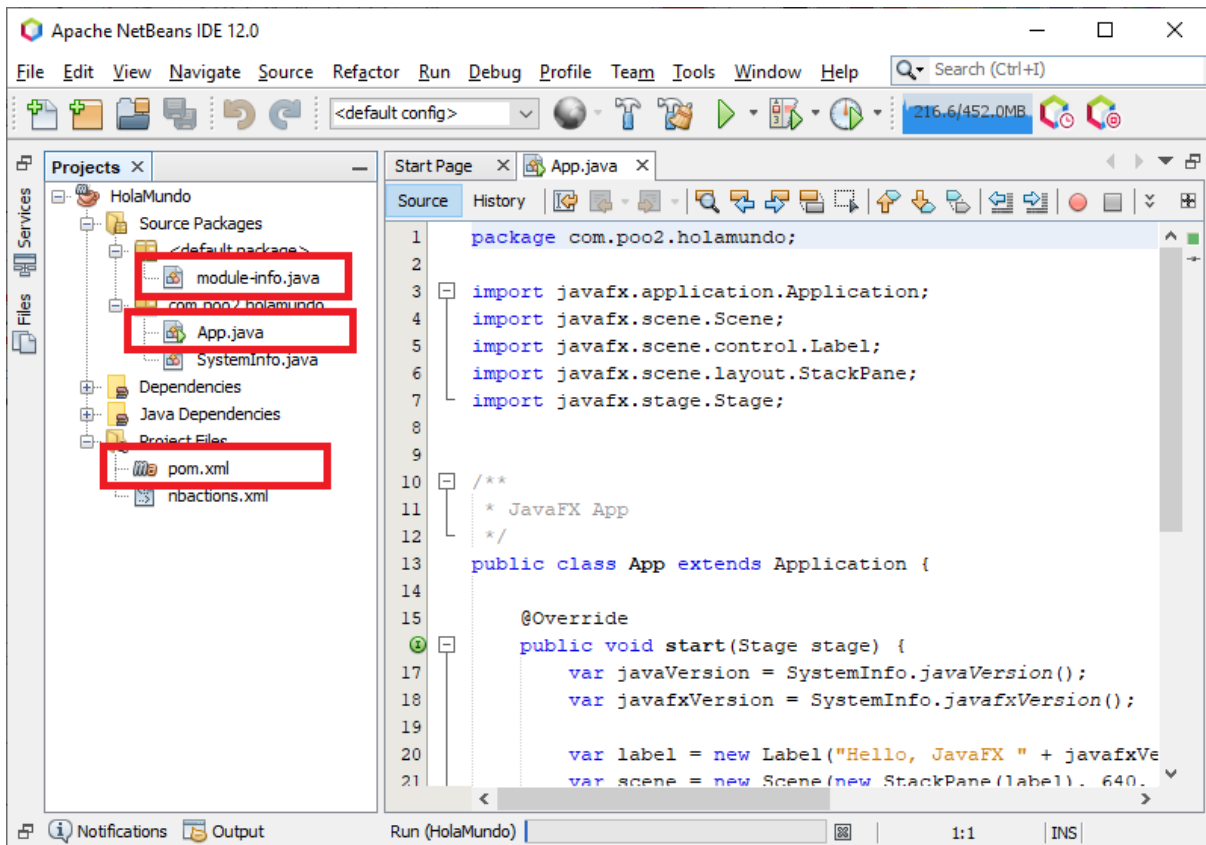


Vamos a ver cómo funciona esto, pero primero vamos a ver la estructura del proyecto en la barra lateral (si no aparece la opción, puede abrir el panel en la barra de herramientas *Window > Project*)



Hay 3 archivos que nos van a interesar

- module-info.java
- App.java
- pom.xml



App.java es la aplicación en sí, contiene el código que se ejecuta pom.xml es el archivo de configuración, tiene la información de cómo construir nuestro proyecto module-info contiene las dependencias del proyecto

Ahora vamos a ver qué contiene cada uno de los archivos

Primero la clase App.

```
public class App extends Application {

    @Override
    public void start(Stage stage) {
        var javaVersion = SystemInfo.javaVersion();
        var javafxVersion = SystemInfo.javafxVersion();

        var label = new Label("Hello, JavaFX " + javafxVersion + ", running on Java " + javaVersion + ".");
        var scene = new Scene(new StackPane(label), 640, 480);
        stage.setScene(scene);
        stage.show();
    }

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



Lo que hace es crear, dinámicamente, el escenario y la escena que va a desplegar. Lo podemos hacer así, pero nosotros vamos a trabajar con un archivo FXML para usar Scene Builder.

La escena que dibuja esta aplicación contiene un solo elemento, que es un Label, es decir una etiqueta que muestra un texto.

El archivo module-info

```
module com.poo2.holamundo {  
    requires javafx.controls;  
    exports com.poo2.holamundo;  
}
```

Nos muestra las dependencias que va a tener el proyecto, por el momento solamente tiene la de javafx.controls, que son los elementos de la interfaz como etiquetas, botones o campos de texto.

Finalmente el POM, el POM es bastante extenso, pero el módulo que nos interesa es el de dependencias.

```
<dependencies>  
    <dependency>  
        <groupId>org.openjfx</groupId>  
        <artifactId>javafx-controls</artifactId>  
        <version>13</version>  
    </dependency>  
</dependencies>
```

Es dentro de este, donde colocamos nuestras librerías.

## 5. Creando “Hola Mundo”

Ahora vamos a ver como agregar dependencias, configurar el proyecto y crearemos un

### ***Hola Mundo***

Continuemos con la aplicación de muestra.

Lo primero que tenemos que hacer es buscar la dependencia de FXML para poder agregar los archivos. Como estamos usando Maven, lo que debemos hacer para encontrar las dependencias es ir a [mvnrepository.com](https://mvnrepository.com), y buscar la librería que nos interesa, en nuestro caso necesitamos la de JavaFX FXML, ya que el proyecto está usando la versión 13 de java fx, busquemos también la versión 13 de fxml. La podemos encontrar [aquí](#)

Para agregar la dependencia al proyecto, tenemos que colocarla dentro del módulo de dependencias del POM de la siguiente forma

```
<dependencies>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>13</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.openjfx/javafx-fxml -->
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>13</version>
  </dependency>
</dependencies>
```

Ahora vamos al archivo module-info y agregamos las siguientes líneas

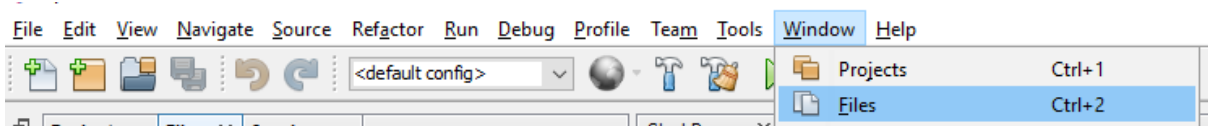
```
module com.poo2.holamundo {
    requires javafx.controls;
    requires javafx.fxml;
    requires java.base;
    exports com.poo2.holamundo;
}
```

*requires javafx.fxml;*

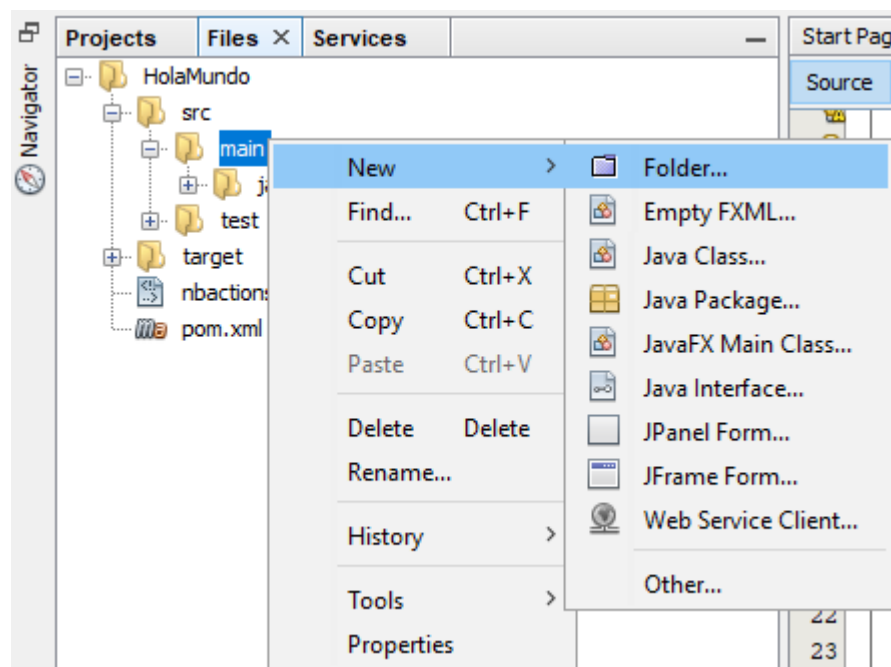
*requires java.base;*

Estos son para poder usar el API de FXML y para poder usar los métodos de java. Ahora podemos agregar la nueva ventana, pero primero, al usar Maven es necesario separar nuestros archivos de código: los .java y los recursos: como imágenes y en este caso, archivos FXML.

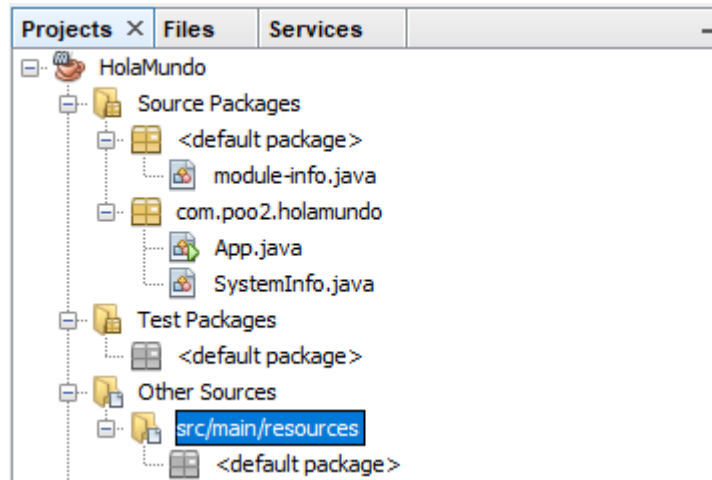
Entonces vamos a la vista de Archivos (si no aparece, puede mostrarla seleccionando *Window > Files*)



Vamos a agregar el directorio *resources*. Luego de la carpeta *main*

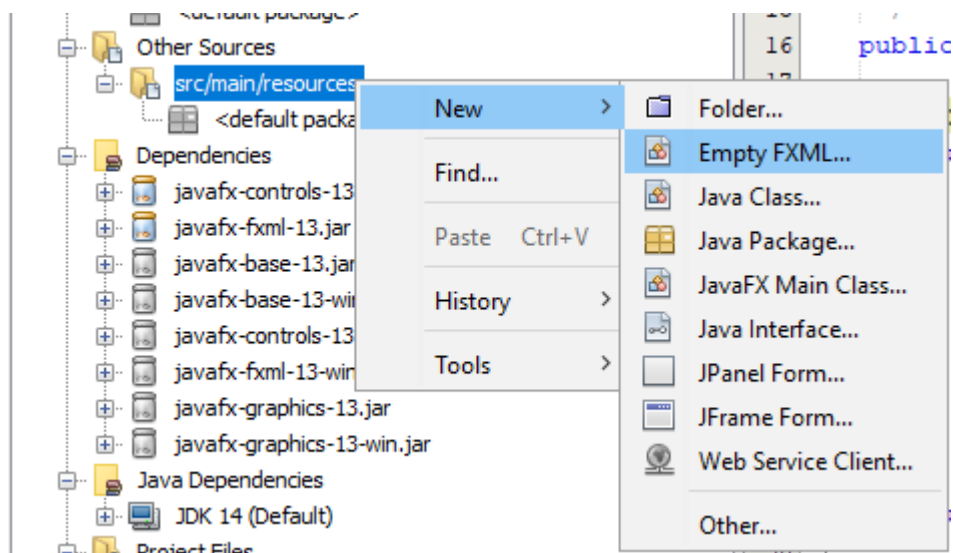


Una vez creado, volvemos a la vista del proyecto, veremos que apareció otro directorio

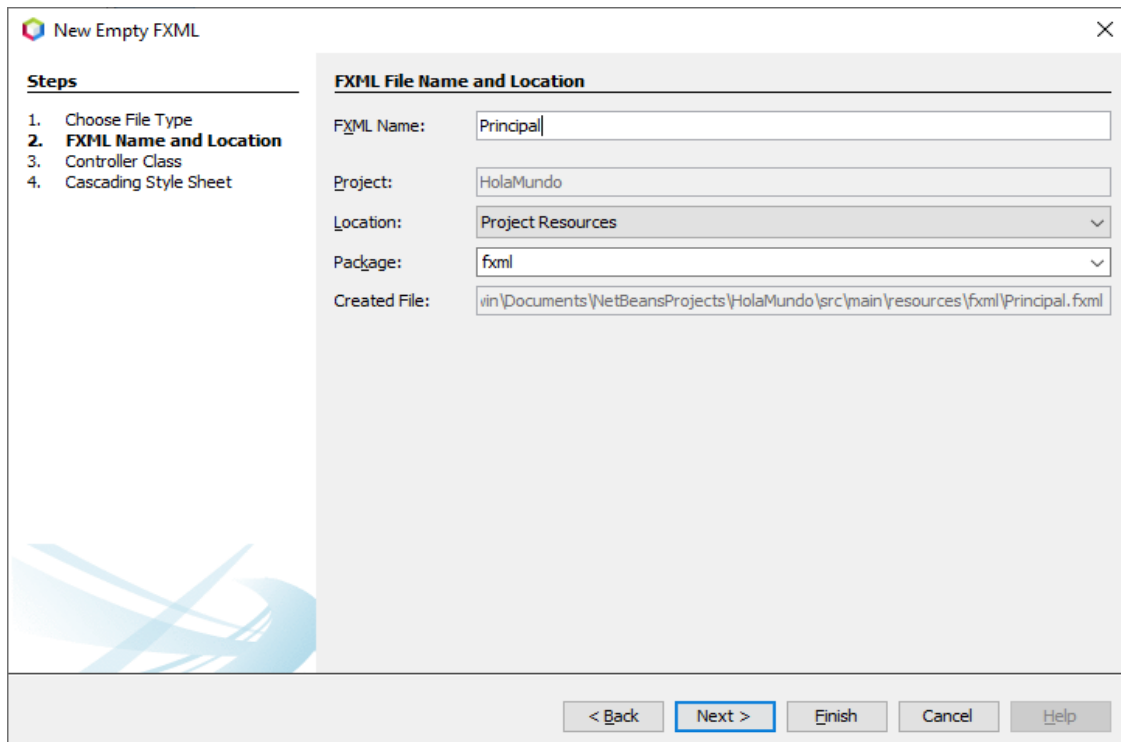


Sobre este es donde debemos agregar nuestros archivos fxml, haciendo clic derecho >

*New > Empty FXML*



Nos mostrará la siguiente ventana



**New Empty FXML**

**Steps**

1. Choose File Type
- 2. FXML Name and Location**
3. Controller Class
4. Cascading Style Sheet

**FXML File Name and Location**

FXML Name:

Project:

Location:

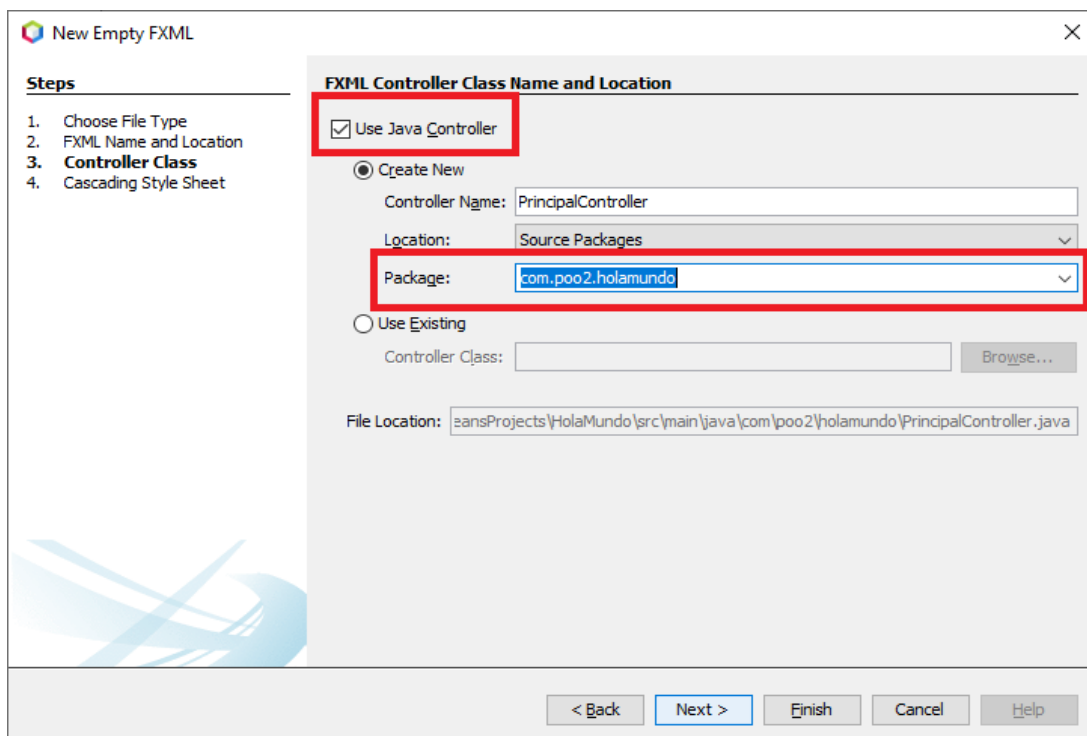
Package:

Created File:

< Back Next > Finish Cancel Help

Como nombre vamos a colocar Principal, luego seleccionamos *Next*

Seleccionamos *Use Java Controller*



**New Empty FXML**

**Steps**

1. Choose File Type
2. FXML Name and Location
- 3. Controller Class**
4. Cascading Style Sheet

**FXML Controller Class Name and Location**

☒ Use Java Controller

☐ Create New

Controller Name:

Location:

Package:

☐ Use Existing

Controller Class:  Browse...

File Location:

< Back Next > Finish Cancel Help

Y si no está seleccionado, también seleccionamos el paquete que sea el de nuestro proyecto. Y ahora seleccionamos *Finish*

Con el archivo creado vamos a cambiar el contenido del archivo App.java para que ahora use el FXML que definimos. Coloquemos el siguiente código en el archivo

```
package com.poo2.holamundo;

import java.io.IOException;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

/**
 * JavaFX App
 */
public class App extends Application {

    @Override

    public void start(Stage stage) throws IOException {

        Parent root =
FXMLLoader.load(getClass().getResource("/fxml/Principal.fxml"));

        Scene scene = new Scene(root);

        stage.setScene(scene);

        stage.show();
    }
}
```

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

Lo que estamos haciendo es obtener una referencia al archivo fxml que hemos creado en la variable root. Este archivo se encuentra en el folder src/main/resources/fxml. Pero el método getResources busca en el folder de resources, por lo que tenemos que indicar que es el folder fxml.

Luego cargamos el fxml a la escena, y finalmente lo mostramos

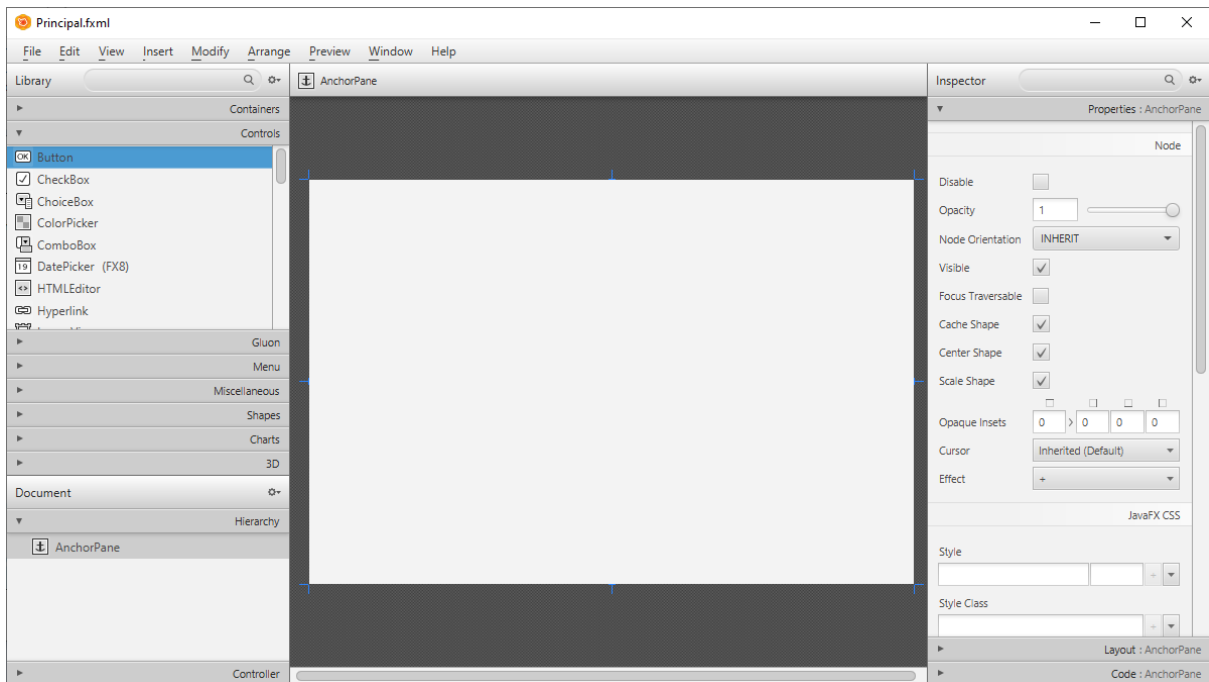
Si ejecutamos el programa ahora nos mostrará una ventana vacía.

### **Creando la interfaz**

Ahora vamos a modificar el fxml para mostrarnos un botón que al presionarlo nos muestre el texto “Hola Mundo”

Entonces para abrir Scene Builder, seleccionamos nuestro archivo *Principal.fxml*, luego

Click Derecho > Abrir, esto abrirá el editor de Scene Builder

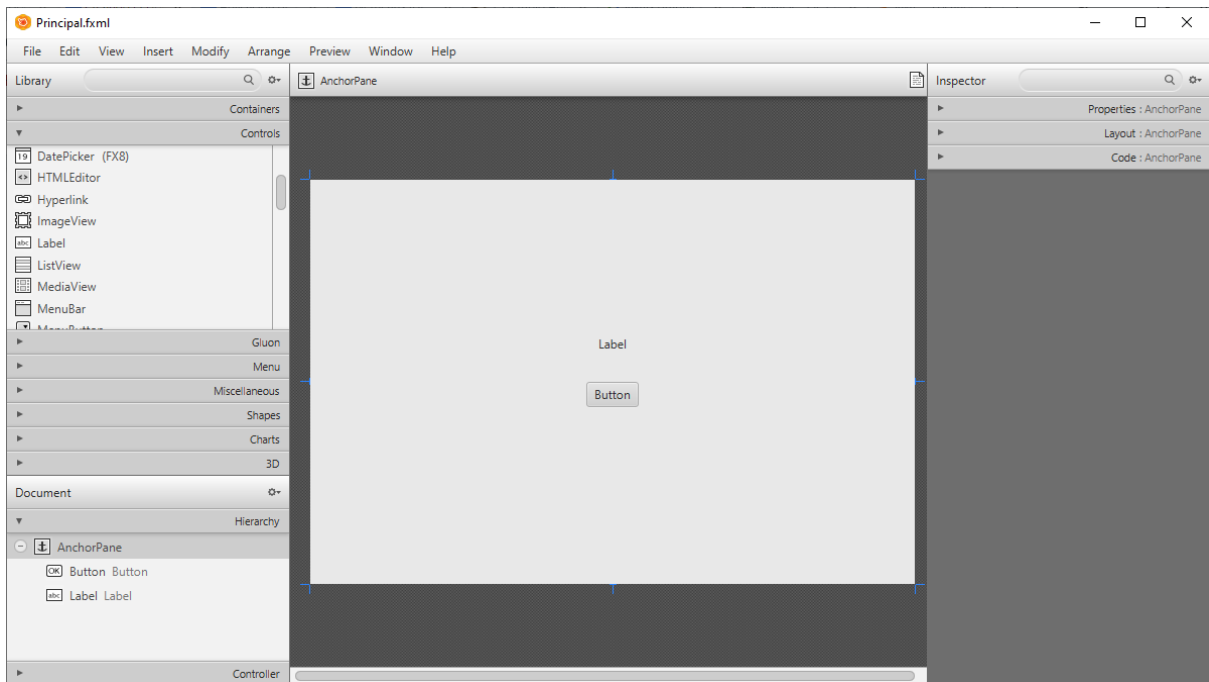


Este es el editor, lo vamos a ver más a fondo en los siguientes videos, pero ahora vamos a colocar un botón y un label.

Del menú izquierdo, la segunda opción que dice Controls, buscamos el que dice Button, lo seleccionamos y arrastramos hasta la posición deseada en el panel del centro.

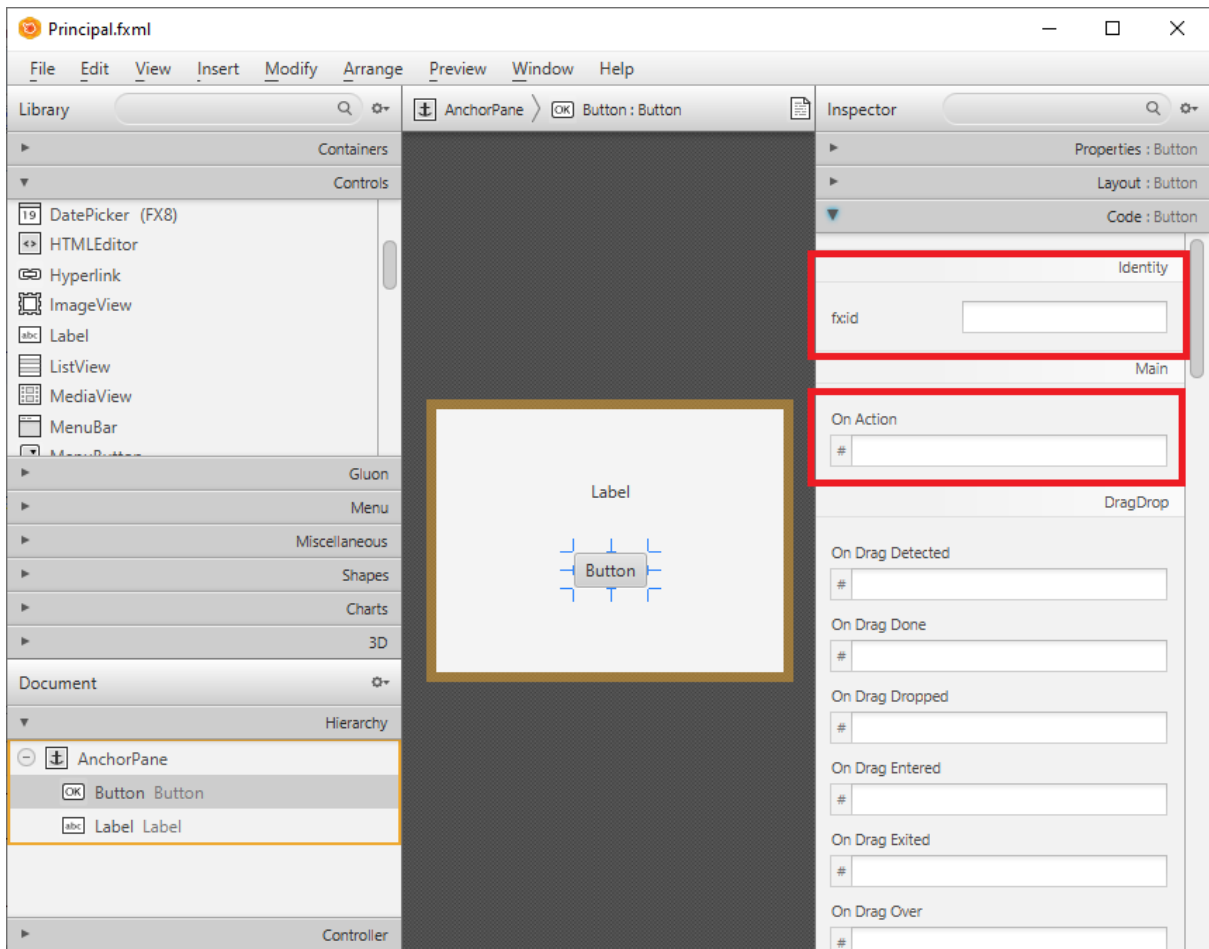
Hacemos lo mismo con el control Label





Si la ventana es muy grande, podemos hacer click en el fondo (que es un Anchorpane), esto mostrará los selectores en las esquinas, y nos permitirá cambiar el tamaño, puede que sea necesario mover el botón y el label luego de cambiar el tamaño

Cuando esto esté listo, seleccionamos el botón, y ahora en el menú derecho seleccionamos la tercera opción: Code



Aquí tenemos que colocar dos valores, fx:id y OnAction.

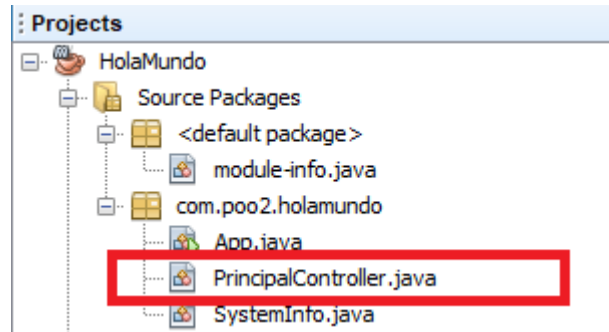
A fx:id, coloquemos el nombre helloButton, y a OnAction colocamos changeLabel

Ahora seleccionemos el label, este no tiene propiedad OnAction, así que solo coloquemos el fx:id, pongámosle de nombre helloLabel

Guardamos los cambios (*File > Save*) o con *Control+S*

La ventana ya está lista, ahora tenemos que colocar la funcionalidad.

Vamos al archivo PrincipalController.java



Y colocamos el siguiente código

```
package com.poo2.holamundo;

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

public class PrincipalController implements Initializable {

    @FXML
    Label helloLabel;

    @FXML
    Button helloButton;

    @FXML

    public void changeLabel(){
```

```
        helloLabel.setText("Hola Mundo");  
    }  
    /**  
     * Initializes the controller class.  
     */  
    @Override  
    public void initialize(URL url, ResourceBundle rb) {  
        // TODO  
    }  
}
```

Lo que estamos haciendo es: la etiqueta @FXML le dice al programa, que la variable está relacionada a un control de la interfaz que acabamos de diseñar. por lo que tienen que tener el mismo nombre que ya les habíamos colocado.

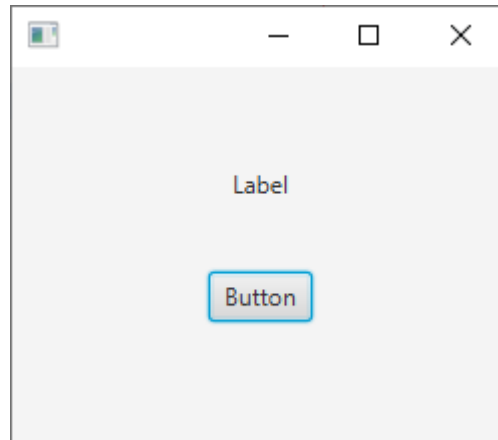
El método changeLabel, lo que hace es colocar el texto que aparece en la label helloLabel, cuando hagamos click en el botón.

Pero antes de ejecutar volvamos a nuestro archivo module-info, y coloquemos la siguiente línea

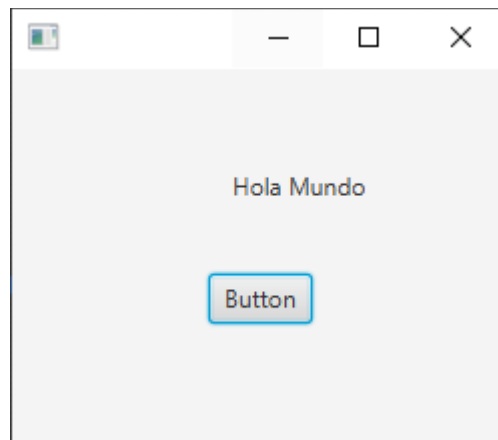
***opens com.poo2.holamundo;***

Eso nos sirve para poder utilizar la reflexión, es decir la forma abreviada de declarar las variables con la etiqueta @FXML

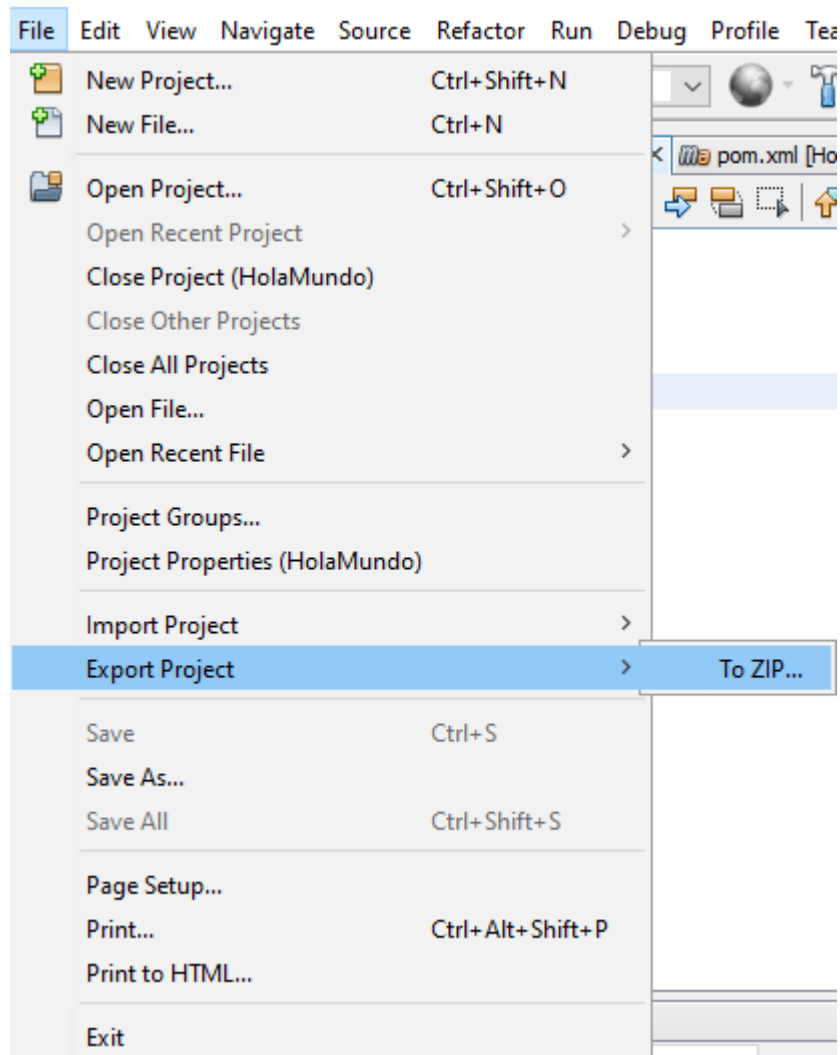
Ahora si podemos ejecutar y nos mostrará la siguiente ventana



Y al hacer clic

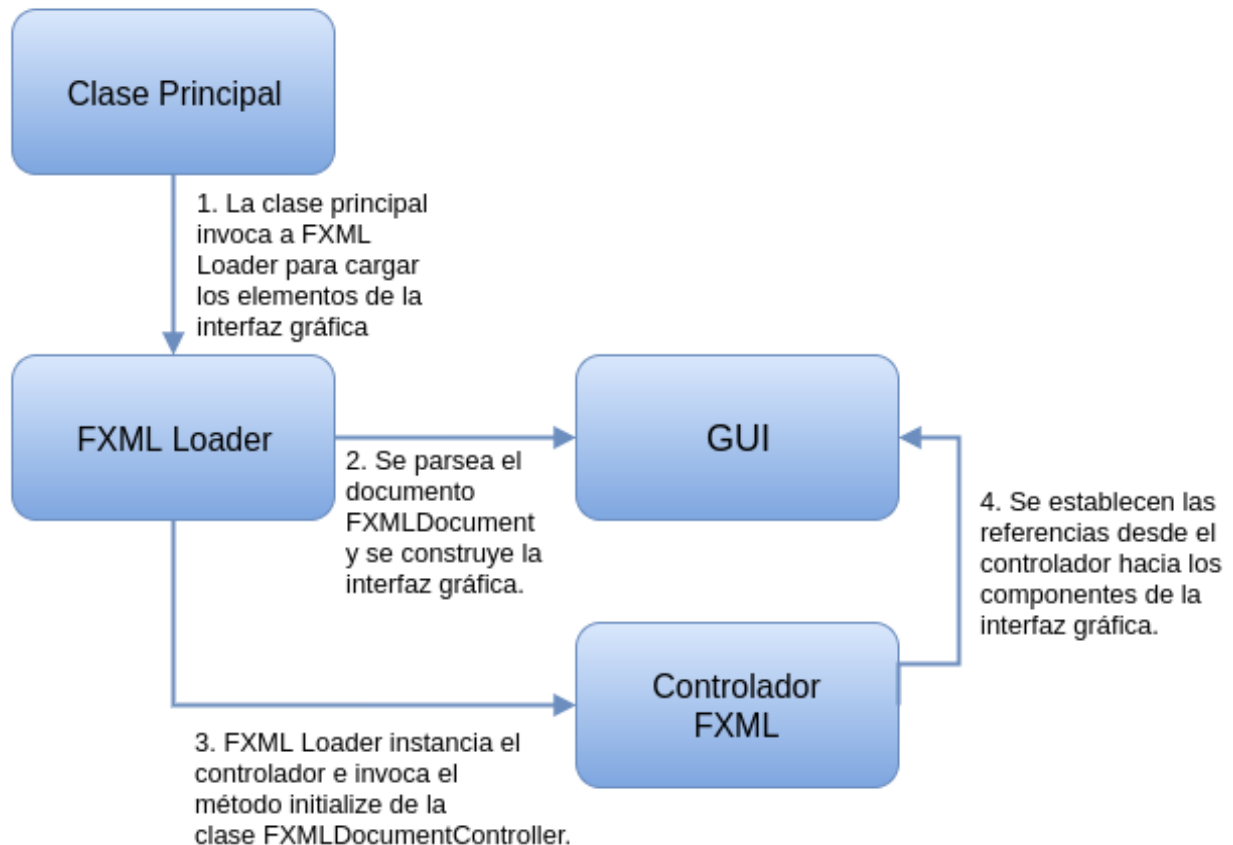


En este punto podemos guardar este proyecto ya configurado y utilizarlo como base para los demás ejercicios que vamos a realizar. Lo podemos hacer como File > Export Project > to ZIP




Y para reutilizarlo el proceso inverso. File > Import Project > from ZIP

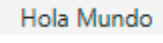

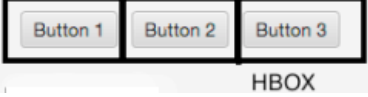


Podemos resumir cómo funciona la aplicación de HolaMundo en JavaFX con el siguiente diagrama



## 6. Elementos de interfaz gráfica en JavaFX

A continuación, vamos a detallar la función de los nodos que se utilizan comúnmente en JavaFX. Las clases que definen estos elementos se encuentran dentro del paquete `javafx.scene.control`. Cada clase contiene los atributos y los métodos que nos permiten manipular los datos dependiendo de las acciones que se realicen con estos elementos.

Nombre	Descripción	Ejemplo gráfico
Button	La acción de un botón es dar clic sobre él. Un botón	

	puede contener texto e imágenes.	
Label	Las etiquetas sirven para mostrar texto dentro de la interfaz.	
TextField	Sirven para que el usuario ingrese texto dentro de la aplicación.	
HBox	Es un contenedor que agrupa elementos de forma horizontal.	
VBox	Es un contenedor que agrupa elementos de forma vertical.	
ImageView	Es un contenedor que permite mostrar una imagen en nuestra aplicación	

Existen muchos tipos de controles que podemos utilizar en JavaFX, desde botones, hasta listas desplegables, tablas y figuras.

Veamos algunos de los más comunes y cómo podemos obtener eventos que nos permitan interactuar con la interfaz.



Para ver los diferentes controles vamos a trabajar con la aplicación de *HolaMundo*, pero vamos a borrar todo el contenido del archivo fxml y del controlador.

## TextField

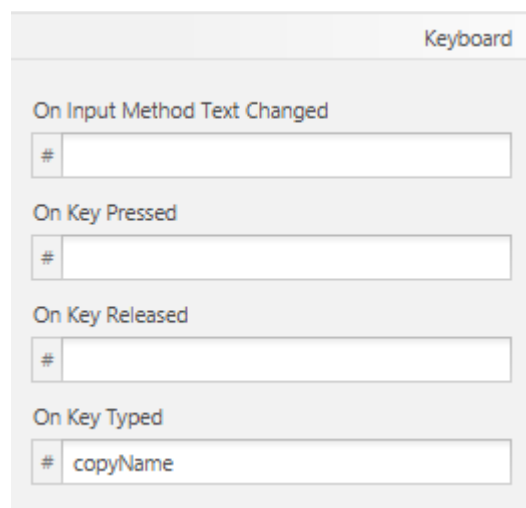
Empecemos con el text field. Es uno de los campos más comunes con los que vamos a tratar. Permite al usuario ingresar un valor.

Para este ejemplo, usaremos un text field en el que el usuario va a ingresar su nombre, y mientras lo escribe vamos a actualizar un label con el nombre que esté ingresando.

Primero coloquemos un Text Field en el Anchorpane, le vamos a dar valores a tres propiedades, primero en el menú de Code el fx:id, le colocamos nameTextField.

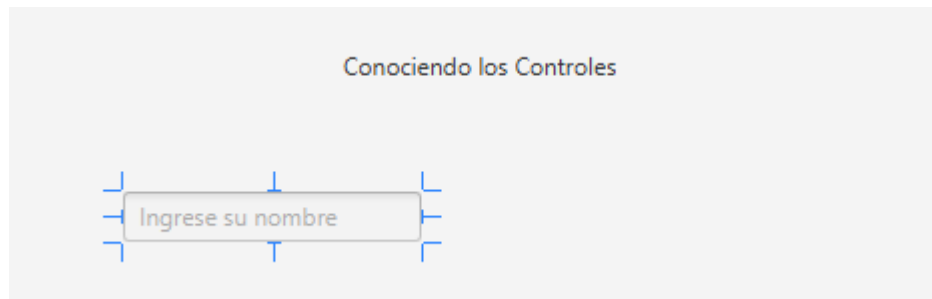
Luego en el menú de Properties, el Text Field tiene una propiedad llamada Prompt Text, esta propiedad muestra una sugerencia de texto que le indica al usuario que debe ingresar, sin colocar texto en el Text Field. Si queremos colocar un valor por defecto, podemos usar la propiedad Text.

La tercera propiedad se encuentra en el menú de Code, es el método On Key Typed



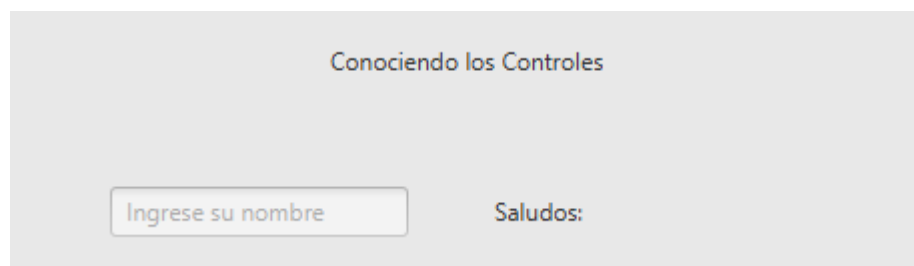
Le daremos el valor copyNombre

Ya con las propiedades, se debería ver así



Ahora agreguemos un Label, éste será para colocar el texto que escribe el usuario, como al TextField le vamos a dar valores a dos propiedades, primero el fx:id, en este caso será labelResultado, y en segundo lugar al campo Text le vamos a dar el valor de “Saludos: “

Lo colocamos cerca del TextField de esta manera



Ahora podemos ir al controlador.

Primero vinculamos los dos controles que acabamos de colocar

```
@FXML
Label nameLabel;

@FXML
TextField nameTextField;
```

Y ahora creamos el método de copiar Nombre de la siguiente forma

@FXML

```
public void copyName(){  
    String name= nameTextField.getText();  
    String result= String.format("Saludos: %s", name);  
    nameLabel.setText(result);  
}
```

Este método se llama cada vez que en el TextField se presiona una tecla, es decir cuando estamos escribiendo.

Lo primero que hace este método es obtener el texto que tiene el TextField con el método `getText()`

Luego usamos una de los métodos de String: `format`, esta función nos permite usar *placeholders* o lugares reservados para incrustar valores dinámicos en nuestros String. Por ejemplo `%s` para reemplazar por un String, `%d` para reemplazar por un entero, o `%f` para reemplazar un número con posiciones decimales. .

El método `format` recibe un String que contenga los *placeholders* y 1 o más valores a sustituir, estos se sustituyen en el orden que se colocan de parámetro.

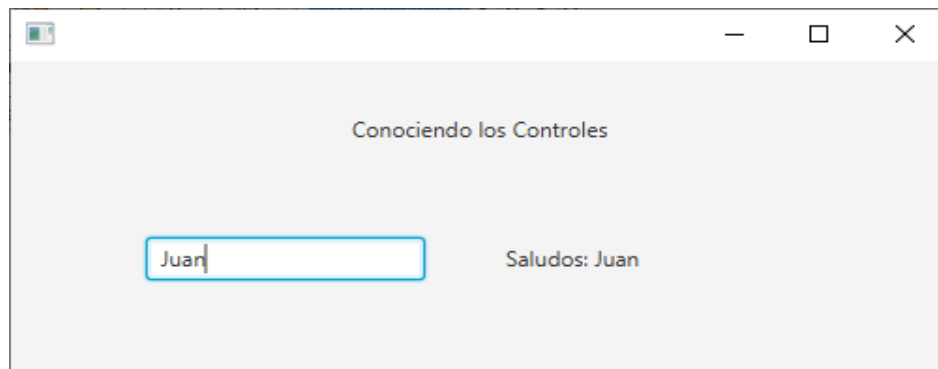
Finalmente lo que hacemos es colocar el valor del String formateado en el Label.

Este código lo podemos resumir de la siguiente forma

@FXML

```
public void copyName(){  
    nameLabel.setText(String.format("Saludos: %s", nameTextField.getText()));  
}
```

Ahora podemos ejecutar el programa, y mientras escribimos veremos que se copia el valor en el Label



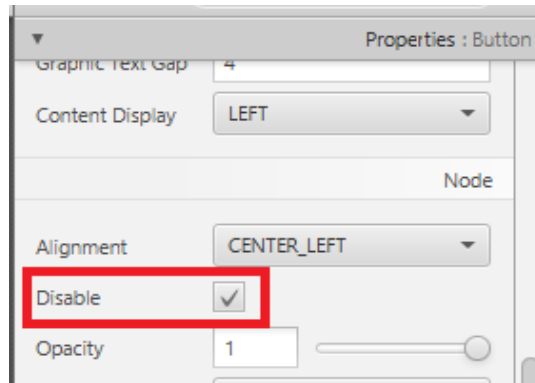
## CheckBox

Ahora veamos la checkbox, es un control bastante sencillo es básicamente un botón que tiene dos estados, seleccionado y no seleccionado, y dependiendo de ese estado podemos cambiar el comportamiento de los demás controles.

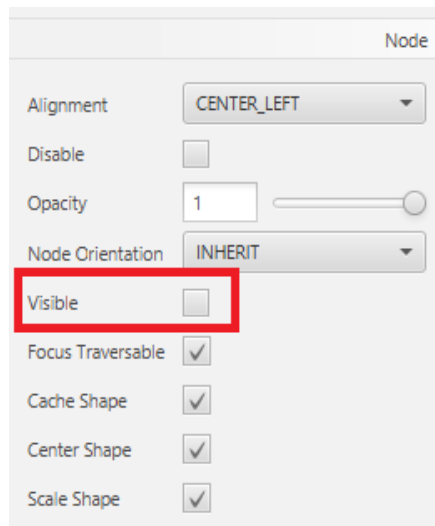
Para esta parte vamos a colocar una CheckBox y dos Botones, como siempre le daremos valor a varias propiedades.

Primero el CheckBox, le colocamos en Text "Habilitar Botones", en fx:id buttonsCheckBox y en el método On Action, changeButtonsState.

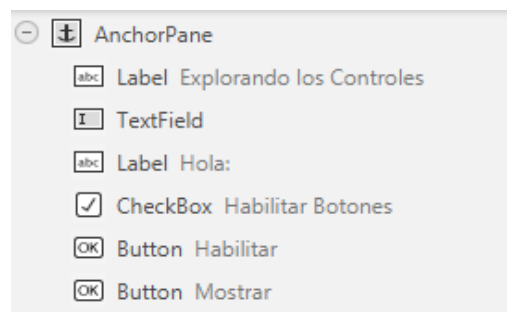
Ahora los botones, el primero le damos el fx:id enableButton, de Text le colocamos Habilitar, y finalmente en el menú de Properties buscamos la opción de Disable y la marcamos.



El segundo botón lo configuramos de manera similar, el fx:id será showButton, de Text colocamos Mostrar, y finalmente en el menú de Properties buscamos el campo de Visible y lo deshabilitamos



**Nota:** Cuando tratamos con objetos que están invisibles por defecto podemos encontrarlos seleccionándolos en el panel de jerarquía



Nos mostrará dónde se encuentra y lo podemos mover y editar



☐ Habilitar Botones

Habilitar



Ahora que tenemos así la interfaz vamos a volver al controlador

Primero enlazamos los controles

```
@FXML
CheckBox buttonsCheckBox;

@FXML
Button enableButton;

@FXML
Button showButton;
```

Luego creamos el método de la siguiente forma

```
@FXML
public void changeButtonsState(){
    if(buttonsCheckBox.isSelected()){
        enableButton.setDisable(false);
        showButton.setVisible(true);
    }else{
        enableButton.setDisable(true);
        showButton.setVisible(false);
    }
}
```

Aquí vamos a manipular el estado de los botones, `setDisable` y `setVisible` funcionan de manera similar, les pasamos un valor Boolean, este valor indica si el control se puede usar o no, y si es visible o no, dependiendo si el valor del Boolean es verdadero o falso respectivamente.

## Sliders

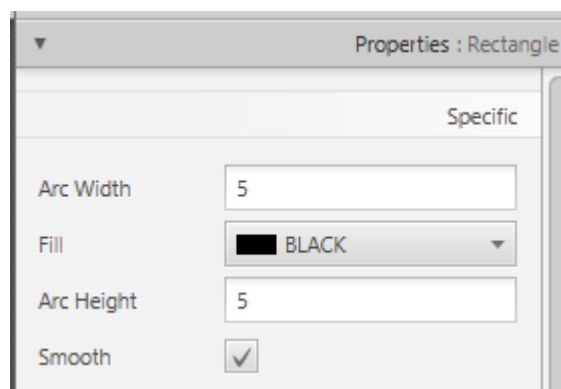
El último control que veremos en esta sección es un slider. Los sliders son controles que permiten seleccionar valores entre un rango por medio de un selector que se arrastra, Son muy útiles para mostrar intensidad, como el brillo de una pantalla o el volumen.

En el menú de Controls buscamos el Slider, y en el menú de Shapes vamos a buscar un rectángulo.

Los colocamos en la ventana, y le daremos algunos valores.

Al slider el `fx:id` le damos el valor de `colorSlider`.

Al rectángulo el `fx:id` le damos el valor de `colorShape`, también al rectángulo, buscamos la propiedad de `fill` y le damos el color negro



Ahora con eso vamos a movernos al Controlador, como siempre enlazamos los controles

```
@FXML
```

```
Slider colorSlider;
```

```
@FXML
```

```
Rectangle colorShape;
```

En este caso vamos a crear un Listener, que es una función que va a estar pendiente de los cambios de valor de nuestro Slider.

En el método de initialize, colocamos el siguiente código

```
colorSlider.valueProperty().addListener(new ChangeListener<Number>() {  
    @Override  
    public void changed(ObservableValue ov, Number oldValue, Number  
newValue) {  
        int value = newValue.intValue()* 255/100;  
        colorShape.setFill(Color.rgb(0, 0, value));  
    }  
});
```

Lo que hace es simple. Coloca un Listener al valor del slider, es decir vamos a estar escuchando constantemente el valor que tiene el Slider y podremos reaccionar cuando cambie.

Declaramos un new ChangeListener ya que queremos una implementación personalizada, y en ella sobrescribimos el método changed, que se ejecuta cuando se detecta un cambio.

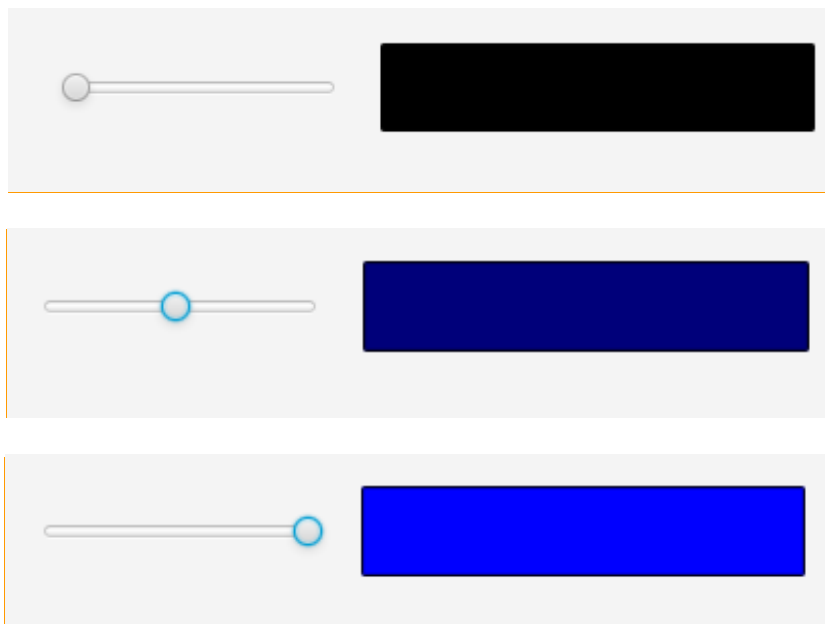


De esta función lo que nos interesa va a ser el tercer parámetro, que es el nuevo valor que tiene el Slider una vez que se ha movido.

Con esto vamos a cambiar el color del rectángulo, esto lo hacemos con el método `setFill` de la figura, y como valor pasamos un color RGB.

Los colores RGB van de 0 hasta 255 en intensidad, mientras que el valor de un Slider va de 0 hasta 100, por lo que tenemos que hacer una equivalencia para colocar el valor como lo esperamos.

Al ejecutar podemos ver que la figura empieza con un color negro, pero mientras movemos el Slider a la derecha, se va volviendo más azul



## 7. Estilos en JavaFX

Los elementos de la interfaz gráfica se pueden personalizar al definir los estilos con CSS.

Los CSS, hojas de estilo en cascada por sus siglas en inglés, nos permiten definir un

conjunto de reglas que definen cómo se van a desplegar los elementos y estas reglas se pueden aplicar jerárquicamente a los nodos.

El formato de una propiedad en CSS es el siguiente:

**nombre\_propiedad: valor\_personalizado**

Las etiquetas de las propiedades de CSS en JavaFX varían del CSS que se utiliza en HTML. A la mayoría de las propiedades se les antepone el prefijo -fx-. Por ejemplo para cambiar el color de fondo de un elemento:

Propiedad background-color en CSS para HTML	Propiedad background-color en CSS para JavaFX
-background-color: blue;	-fx-background-color: blue;

Algunas de las propiedades más utilizadas que se pueden utilizar en JavaFX son las siguientes:

Descripción	Propiedad CSS	Valores posibles	Ejemplo
Cambiar tamaño de texto	-fx-font-size	<p>Valor numérico acompañado de la unidad de medida.</p> <p>La unidad de medida puede ser relativa o absoluta. Si es relativa se utiliza % (porcentaje). Si es absoluta pueden ser px (pixel), em</p>	<p>-fx-font-size: 30%;</p> <p>-fx-font-size: 15px;</p> <p>-fx-font-size: 12pt;</p> <p>-fx-font-size: 12em;</p>

		(depende de la fuente) o pt (punto).	
Cambiar fuente de texto	-fx-font-family	El nombre de la fuente. La fuente indicada debe estar disponible en el sistema.	-fx-font-family:Arial;
Estilo de texto Negrita	-fx-font-weight	Puede ser una de las siguientes opciones: normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800 o 900	-fx-font-weight:bold;
Color de texto	-fx-text-inner-color	El nombre del color en inglés o su representación en hex.	-fx-text-inner-color:red;  -fx-text-inner-color:#FF0000;
Color de fondo de un elemento	-fx-background-color	El nombre del color en inglés o su representación en hex.	-fx-background-color: black;  -fx-background-color: #000000:
Cambiar color por defecto de elemento	-fx-base	El nombre del color en inglés o su representación en hex.	-fx-base: #b6e7c9;

## Agregar estilo directamente a un elemento

Para agregar estilo directamente a un elemento se puede hacer de dos formas:

### Desde código

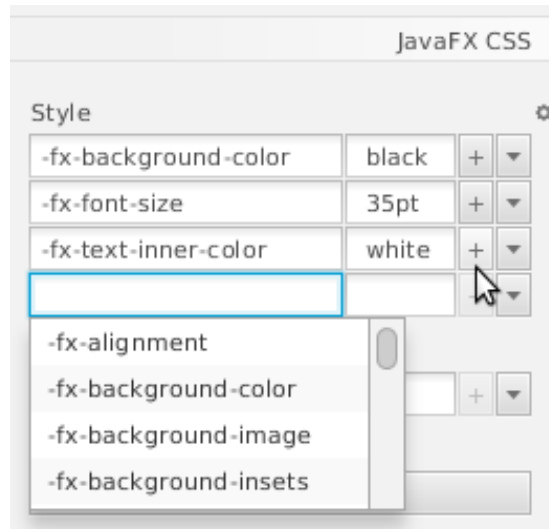
```
Button ejemplo = new Button("Ejemplo");  
  
ejemplo.setStyle("-fx-background-color: red;");
```

## En Scene Builder

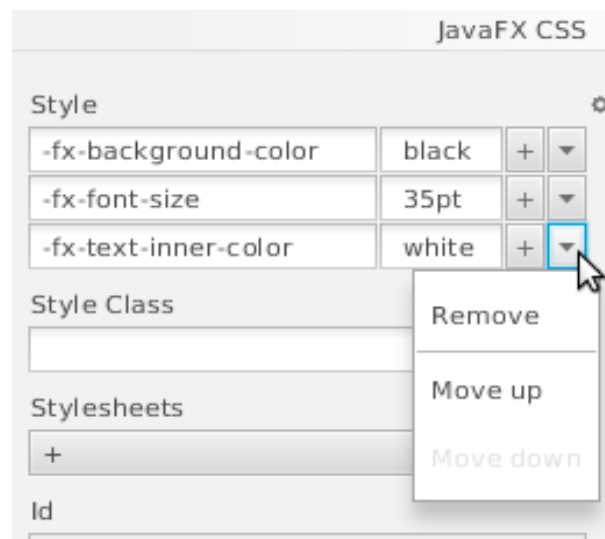
En el lado derecho dentro del inspector en la pestaña de propiedades buscamos la sección de JavaFX CSS.



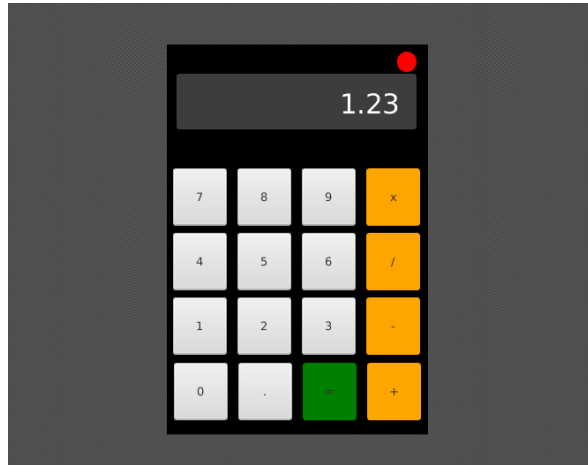
Al dar clic en + podemos agregar nuevos estilos. En el primer input se coloca el nombre de la propiedad y en el segundo en valor:



Al dar clic en la flecha podemos eliminar la propiedad o cambiar el orden:



Por ejemplo utilizando los elementos vistos anteriormente y los estilos podemos crear interfaces como la siguiente:



## Agregar estilos utilizando id y clases de CSS

Es habitual que cuando se usan muchas reglas o propiedades de CSS se agrupen y se definan una sola vez para varios elementos o se agrupen para un solo elemento. Para ello CSS nos provee varios selectores entre ellos se encuentran el selector por id y el selector por clase. Los selectores sirven para hacer referencia a un elemento por medio de un id o a un conjunto de elementos que pertenecen a una clase. El selector de id se declara anteponiendo el símbolo de numeral al identificador del elemento y luego se listan las propiedades entre llaves. Por ejemplo:

```
#boton {  
  -fx-background-color: #ff0000;  
  -fx-text-fill: #ffffff;  
}
```

El selector de clase se declara anteponiendo un punto al nombre de la clase y luego se listan las propiedades entre llaves. Por ejemplo:

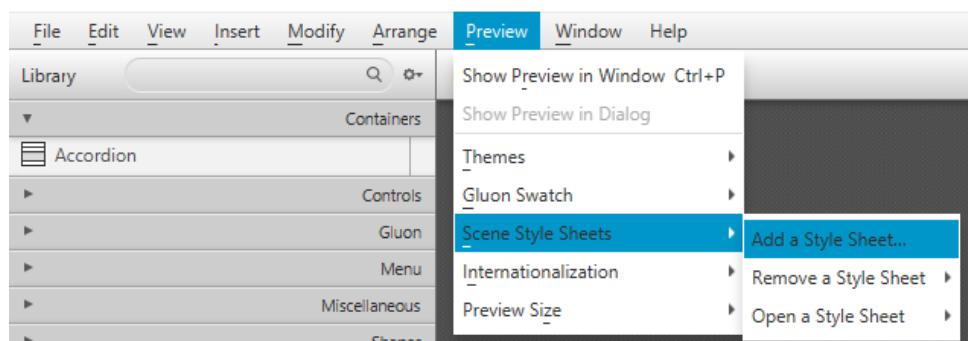
```
.botones {  
  
    -fx-background-color: blue;  
  
    -fx-text-fill: #ffffff;  
  
}
```

De esta forma todos los elementos que pertenezcan a la clase botones serán de color azul y tendrán color de texto blanco.

## Utilizando selectores desde Scene Builder

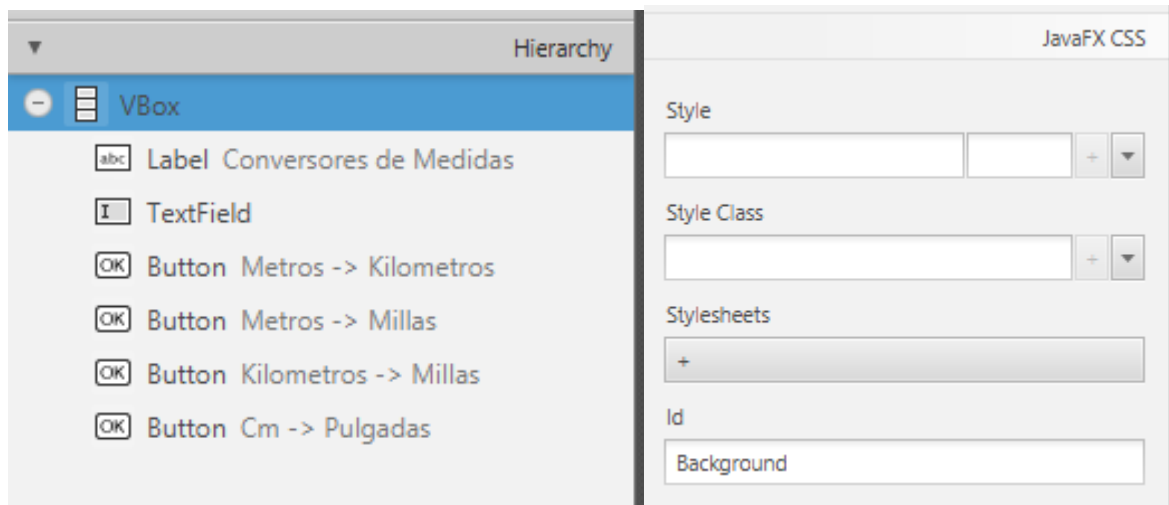
Para utilizar id y clases de CSS en JavaFX se deben listar todos los id y clases en un archivo con extensión .css.

Luego se deben importar desde SceneBuilder dando clic sobre el botón del menú “Preview”, seleccionando la opción "Scene Style Sheets" y dando clic sobre "Add a Style Sheet".



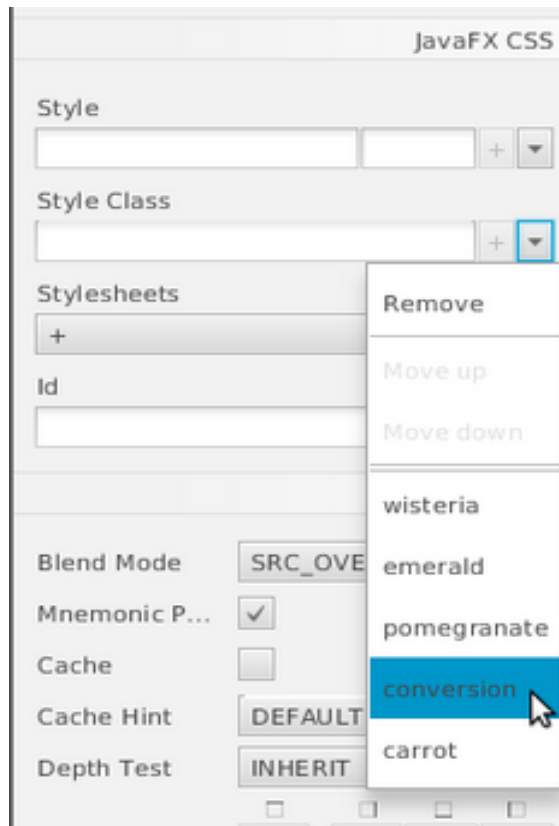
En la ventana de selección de archivo, deben de seleccionar el archivo .css que fue creado anteriormente.

Esto sirve solamente para la vista previa de las vistas, para agregar definitivamente una hoja de estilos a un archivo hay que colocarla en el componente base de nuestra ventana, en el menú lateral derecho, en Properties, la opción de Stylesheet



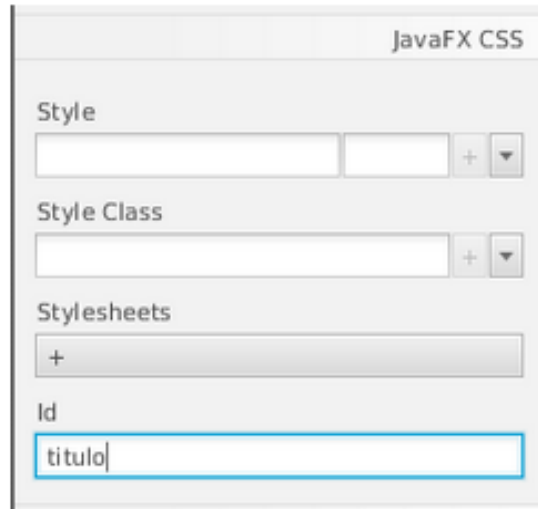
En el menú lateral derecho en la pestaña de propiedades buscamos la sección de JavaFX CSS. Para agregar una clase a un componente se debe dar clic sobre la flecha de Style Class y seleccionar la clase del componente.





De la misma forma que con las propiedades de CSS se puede agregar clases dando click sobre +. Al dar clic en la flecha podemos eliminar la clase o cambiar el orden de la clase.

Para agregar un id a un componente, en la sección de JavaFX CSS se debe ingresar el nombre en el campo de texto Id.



Es importante que el id sea único en la interfaz para que no se tengan conflictos al momento de utilizarlos en el controlador.

Por ejemplo, para crear la siguiente ventana:



Se definieron las siguientes reglas CSS:

```
#AnchorPane {  
    -fx-background-color: #2c3e50;  
}
```

```
#titulo {
  -fx-text-fill: #ffffff; -fx-font-size: 18px;
}
.conversion {
  -fx-font-size: 14px;
  -fx-text-fill: #ffffff;
  -fx-border-color: #ffffff;
  -fx-border-width: 2px;
  -fx-border-radius: 0;
  -fx-border-style: solid;
}
.emerald {
  -fx-background-color: #27ae60;
}
.carrot {
  -fx-background-color: #e67e22;
}
.wisteria {
  -fx-background-color: #8e44ad;
}
.pomegranate {
  -fx-background-color: #c0392b;
}
```

Vamos a ver cómo cambiar los estilos de los componentes, primero como propiedades individuales para cada elemento y luego veremos cómo usar los archivos CSS.

Vamos a crear una aplicación que nos permita realizar conversión entre tipos de medidas.

Lo primero que vamos a hacer es descargar el proyecto base que tiene 4 botones. Al principio se ve así:

Conversores de Medidas

Metros -> Kilometros

Metros -> Millas

Kilometros -> Millas

Cm -> Pulgadas

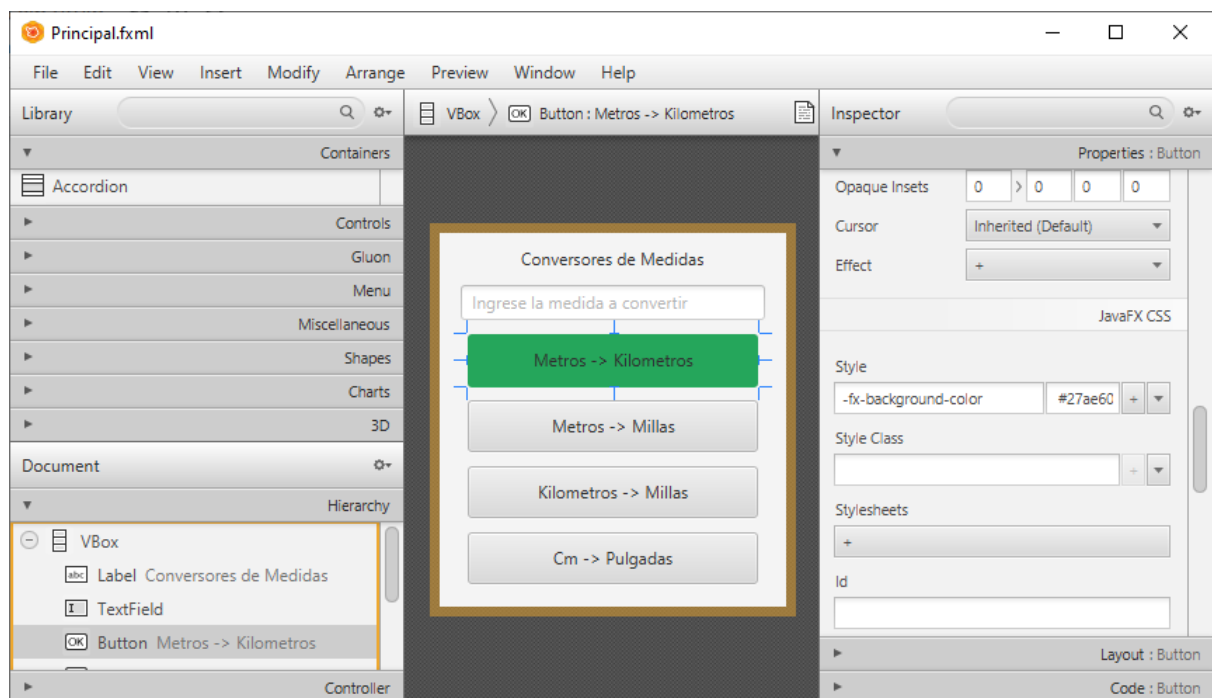
Lo que vamos a hacer es aplicar estilos en los elementos para cambiar como se ve, y luego vamos a usar alertas para mostrar el resultado de la conversión de medidas.

Vamos a utilizar las reglas que estaban definidas anteriormente, con algunas modificaciones, se vería de la siguiente forma:

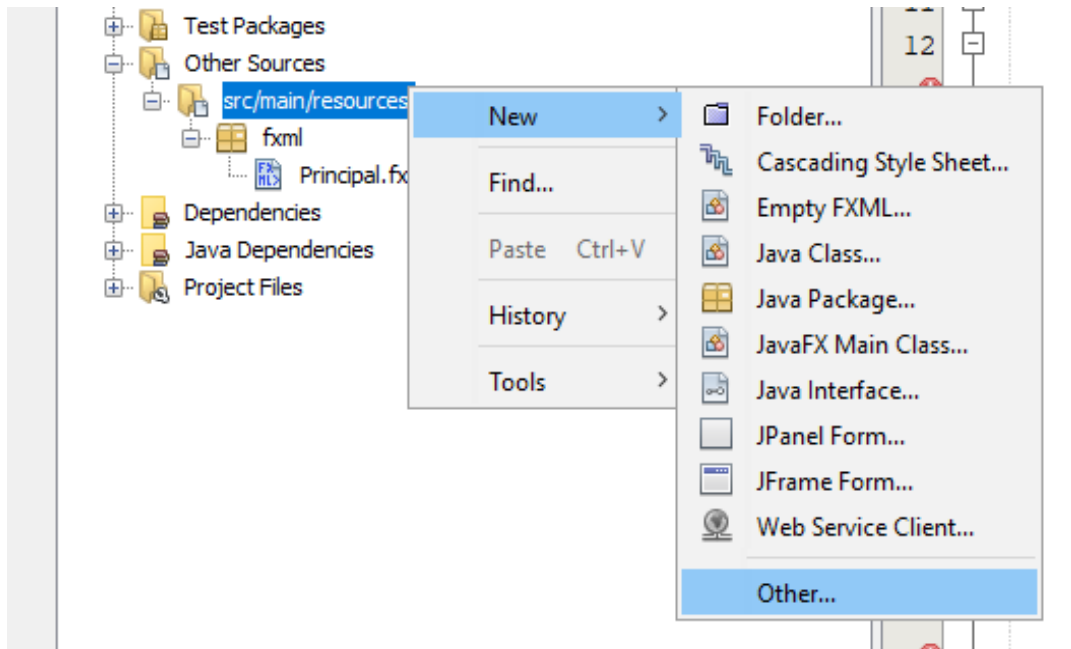
```
#Background {  
  -fx-background-color: #2c3e50;  
}  
#titulo {  
  -fx-text-fill: #ffffff; -fx-font-size: 13px;  
}  
.conversion {  
  -fx-font-size: 14px;  
  -fx-text-fill: #ffffff;  
  -fx-border-color: #ffffff;  
  -fx-border-width: 2px;  
  -fx-border-radius: 0;  
  -fx-border-style: solid;  
}  
.emerald {  
  -fx-background-color: #27ae60;  
}  
.carrot {  
  -fx-background-color: #e67e22;
```

```
}  
.wisteria {  
  -fx-background-color: #8e44ad;  
}  
.pomegranate {  
  -fx-background-color: #c0392b;  
}
```

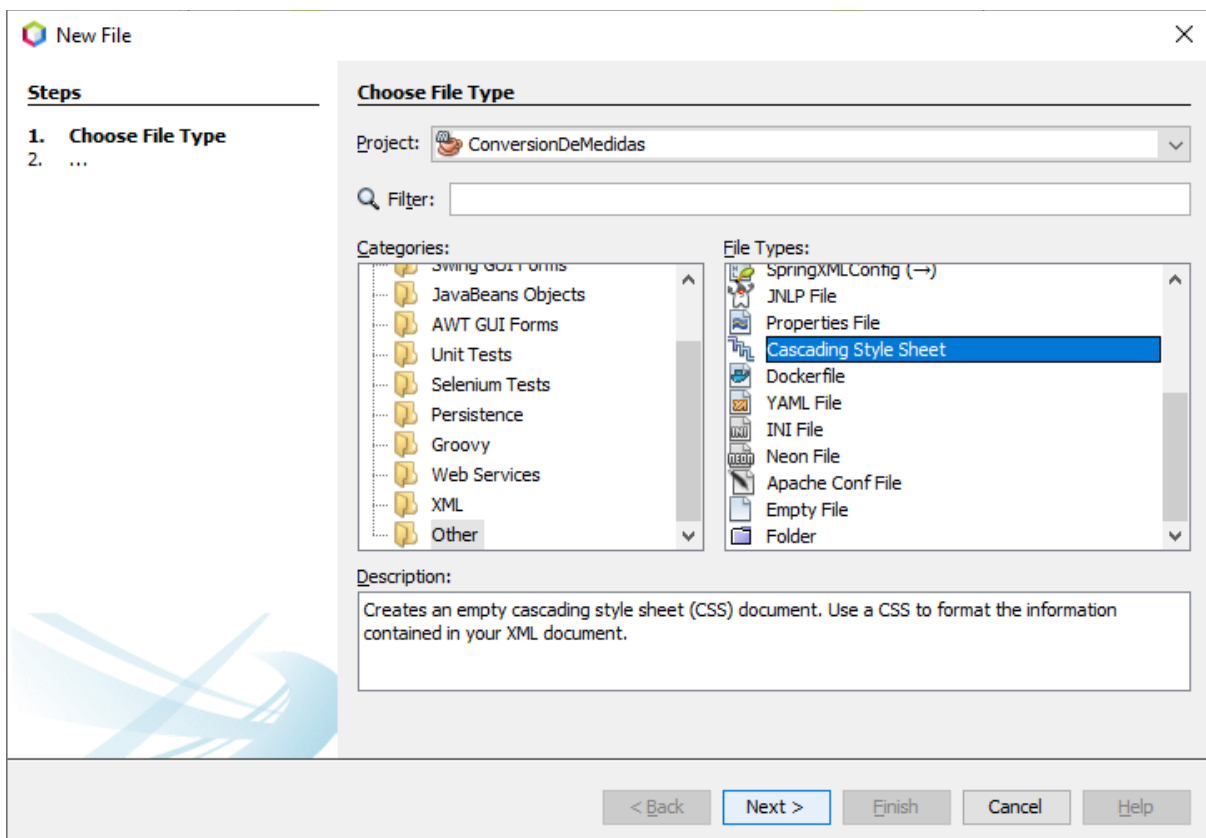
Recordemos que si quisiéramos agregar las propiedades una por una en el editor, podemos seleccionar un elemento y luego en el panel de la derecha, en el menú de Properties colocar la propiedad.



Pero lo que vamos a hacer es usar una hoja de estilos, para eso vamos a agregar una hoja en nuestro paquete de recursos



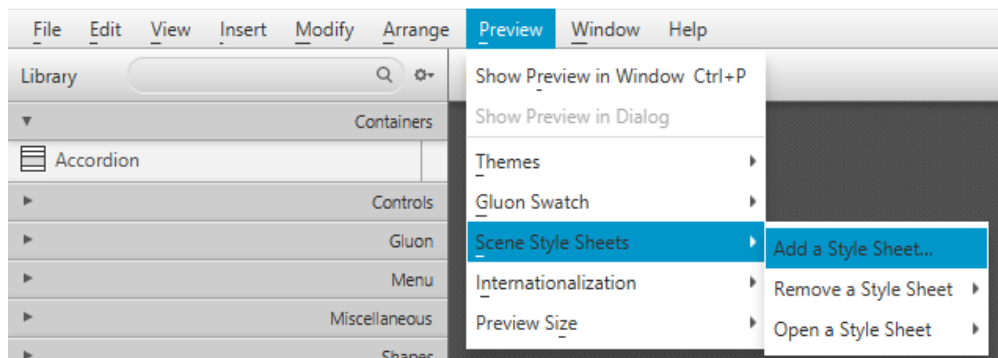
Luego elegimos la categoría *Other* y finalmente la Cascading Style Sheet



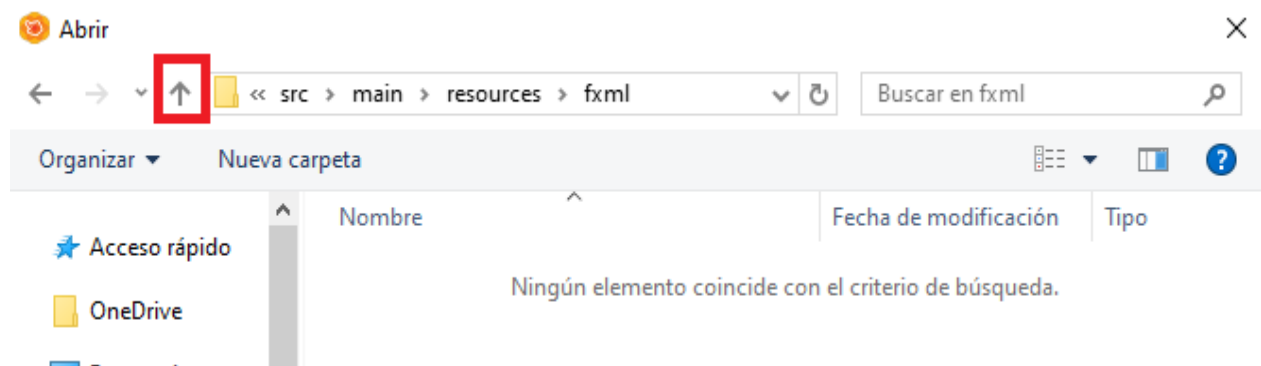
Le damos un nombre, por ejemplo conversión, y hacemos click en *Finish*.

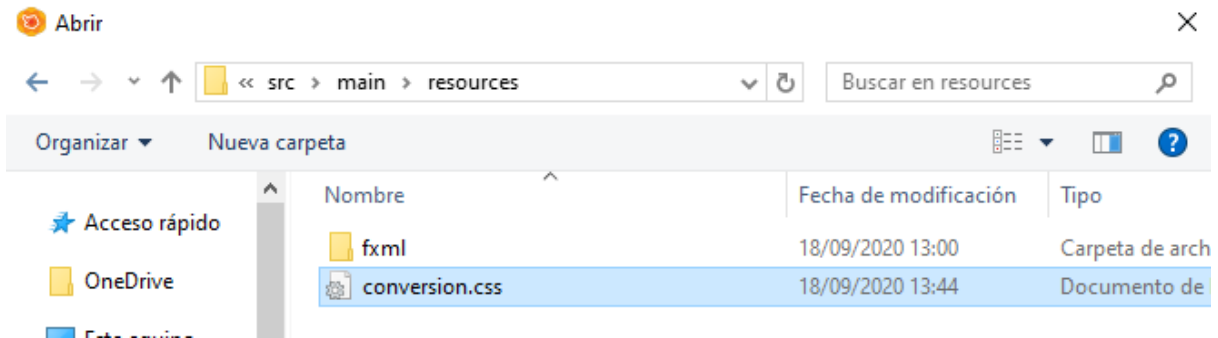
Ahora abrimos la hoja y pegamos los estilos que definimos anteriormente, guardamos los cambios y podemos volver a Scene Builder.

Para poder visualizar como se verá la hoja la agregamos en el preview.



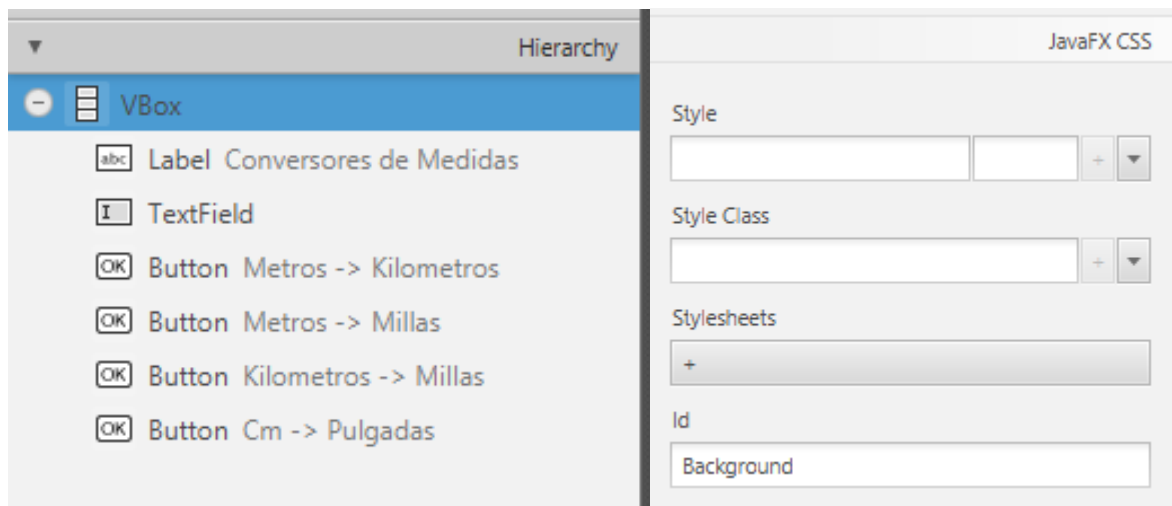
La hoja se encuentra en una carpeta arriba de la carpeta de fxml es decir en el directorio de resources





Seleccionamos la hoja y ahora podemos agregar las clases a nuestros elementos

Por ejemplo seleccionamos la VBox, y luego le colocamos el id Background



Luego le colocaremos a los elementos las siguientes clases

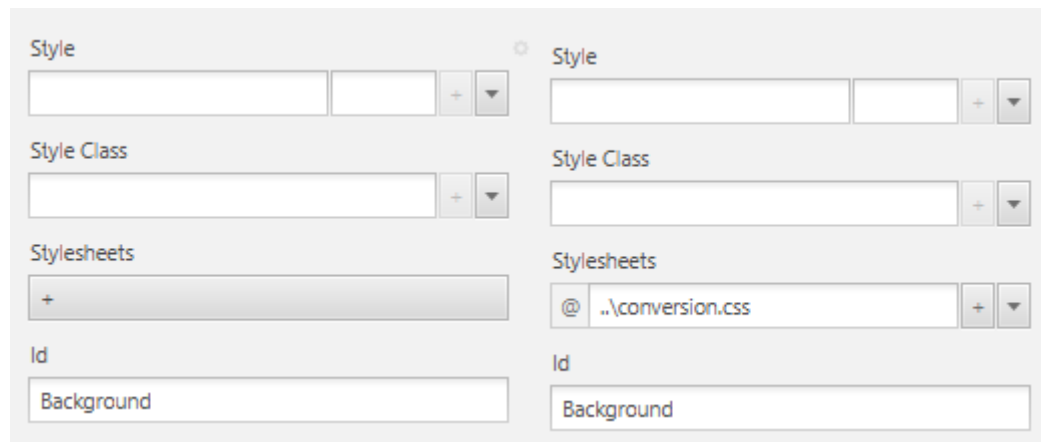
- Al Label, le colocamos el id: titulo
- Al Botón 1 le colocamos las clases conversion y emerald
- Al Botón 2 le colocamos las clases conversion y wisteria
- Al Botón 3 le colocamos las clases conversion y carrot
- Al Botón 4 le colocamos las clases conversion y pomegranate



Una vez hecho esto habrá necesidad de redimensionar la altura del VBox, y una vez terminado se verá así



¿Pero qué pasa ahora? Si ejecutamos la aplicación, no se verá aplicado ninguno de los cambios. Esto es porque colocamos la hoja en Preview, es decir la estamos usando como muestra. Para que se pueda utilizar la hoja en la aplicación la tenemos que agregar a nuestro elemento de fondo, en este caso será al VBox. Para eso vamos al menú de Properties, y agregamos la opción de StyleSheet



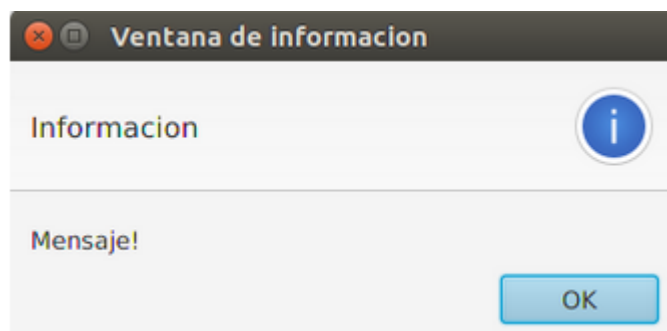
Ya con eso terminamos de aplicar los estilos en nuestra aplicación por medio de un archivo CSS.

## 8. Ventanas de alertas y diálogo

La librería de JavaFX incluye algunas ventanas simples de alertas y diálogo. Para implementarlas puede utilizar el siguiente código:

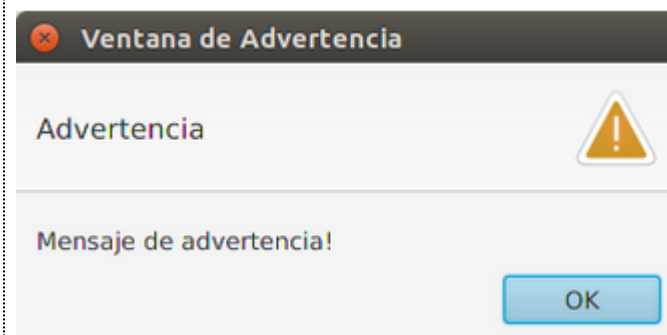
### Ventana de información

```
Alert alert = new  
Alert(AlertType.INFORMATION);  
  
alert.setTitle("Ventana de  
informacion");  
  
alert.setHeaderText("Informacion");  
  
alert.setContentText("Mensaje!");  
  
alert.showAndWait();
```



### Ventana de advertencia

```
Alert alert = new  
Alert(AlertType.WARNING);  
  
alert.setTitle("Ventana de  
Advertencia");  
  
alert.setHeaderText("Advertencia");  
  
alert.setContentText("Mensaje de  
advertencia!");
```



```
alert.showAndWait();
```

## Ventana de error

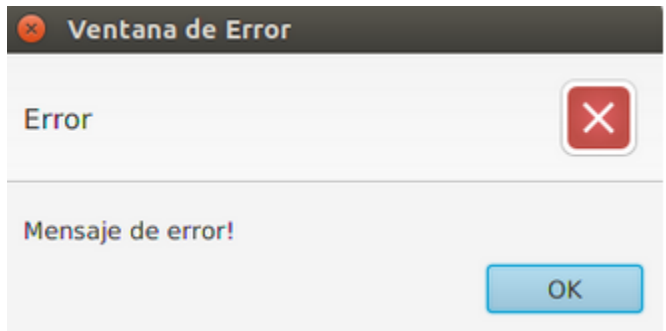
```
Alert alert = new
Alert(AlertType.ERROR);

alert.setTitle("Ventana de Error");

alert.setHeaderText("Error");

alert.setContentText("Mensaje de error!");

alert.showAndWait();
```



## Ventana de confirmación

```
Alert alert = new
Alert(AlertType.CONFIRMATION);

alert.setTitle("Ventana de confirmacion");

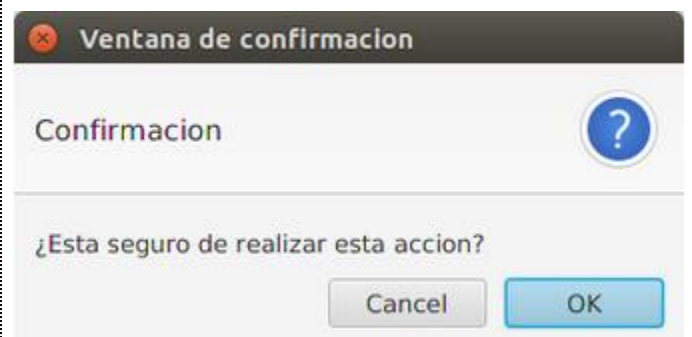
alert.setHeaderText("Confirmacion");

alert.setContentText("¿Está seguro de
realizar esta accion?");

Optional<ButtonType> result =
alert.showAndWait();

if (result.get() == ButtonType.OK) {

    //Instrucciones al dar confirmar la
    accion
```



```
System.out.println("Confirmacion:
OK");

} else {

    //Instrucciones al cancelar la accion

    System.out.println("Confirmacion:
Cancelar");

}
```

Ahora con el proyecto de las conversiones de medidas vamos a agregar funcionalidad a cada uno de los botones. Para eso vamos a mostrar los resultados de la conversión en una ventana de alerta o vamos a mostrar una ventana de error si el campo para ingresar el dato está vacío o no es un número.

Ahora movámonos a nuestro Controlador, en él ya están las funciones que se llaman cada vez que se hace clic en un botón.

Si hacemos clic en un botón como está el proyecto, lo único que hace es mostrarnos en la consola un mensaje que indica qué botón fue presionado.

Entonces lo que vamos a hacer es crear dos funciones más que nos ayuden a mostrar los resultados de las conversiones.

Primero vamos a declarar unas constantes.

```
final double METRO_KILOMETRO = 0.001;

final double METRO_MILLA = 0.00062137;

final double KILOMETRO_MILLA = 0.62137;

final double CENTIMETRO_PULGADA = 0.393701;
```

Estas son las constantes que usaremos para convertir el valor de ingreso, las podemos colocar en cualquier lugar del controlador, pero lo preferible es justo después de la declaración de la clase así:

```
public class PrincipalController implements Initializable {  
  
    final double METRO_KILOMETRO = 0.001;  
    final double METRO_MILLA = 0.00062137;  
    final double KILOMETRO_MILLA = 0.62137;  
    final double CENTIMETRO_PULGADA = 0.393701;  
  
    ...  
}
```

Ahora vamos a crear un método que nos permita verificar si el valor que ingresamos es un número o no.

```
public Integer obtenerValor(){  
    Integer valor = null;  
    try{  
        valor = Integer.parseInt(cantidad.getText());  
    }catch(NumberFormatException e){  
        e.printStackTrace();  
    }  
    return valor;  
}
```

Lo que hace este método es *tratar* de obtener el valor que tiene actualmente nuestro TextField, pero si se da el caso de que lo que contiene el campo no es un número, nos va a dar un error, por eso colocamos la conversión dentro de un *try*. Esta instrucción nos permite ejecutar código y que, si ocurre un error, no se detenga la ejecución. La instrucción que le sigue *catch* nos permite ejecutar código basados en el error que pudiera ocurrir.

La razón por la que usamos Integer y no int es para poder retornar un valor null, en caso que haya un error, no queremos retornar un valor por defecto, ya que puede causar una ejecución incorrecta. Vamos a usar null para poder detectar un error y mostrar una ventana de error cuando sea necesario.

Entonces lo que vamos a hacer es primero verificar el valor que está ingresado en el campo, y si de hecho es un número vamos a mostrar una ventana de alerta con el resultado de la conversión.

Entonces vamos primero con el botón de metros a kilómetros, y reemplazamos el println por este código

```
Integer valor = obtenerValor();  
if(valor != null){  
    double resultado = valor * METRO_A_KILOMETRO;  
    Alert alert = new Alert(AlertType.INFORMATION);  
    alert.setTitle("Resultado de la Conversión");  
    alert.setHeaderText("Conversion Exitosa");  
    alert.setContentText(  
        String.format("El resultado de la conversion de Metros a Kilometros es: %.2f",  
            resultado));
```

```
        alert.showAndWait();  
    }
```

Lo que hacemos acá es obtener el valor del TextField con el método que definimos antes, y si no retorno null (es decir si es un número) mostramos una ventana de información.

El resultado lo obtenemos multiplicando el valor obtenido por una de las constantes que definimos anteriormente.

La ventana tiene de contenido otra función, el String.format que ya hemos mencionado antes. En este caso vamos a reemplazar el resultado de la conversión.

El resultado (como es un double) se reemplaza con el *placeholder* %f.

**Nota:** Cuando reemplazamos un valor decimal como float o double, podemos indicar la cantidad de posiciones decimales colocando un punto y el número de posiciones en medio del % y la f, por ejemplo %.2f indica que el resultado se va a imprimir con 2 posiciones decimales.

Ahora vamos a mostrar una ventana de error cuando lo que fue ingresado no es un número, para eso colocaremos este código en la parte del else

```
    else{  
        Alert alert = new Alert(AlertType.ERROR);  
        alert.setTitle("Ventana de Error");  
        alert.setHeaderText("Error");  
        alert.setContentText("Debe ingresar un numero");  
        alert.showAndWait();  
    }
```

Aquí simplemente mostramos una alerta indicando que ingrese un número.

El código de la función se debería ver así

```
public void metrosKilometros(){
    Integer valor = obtenerValor();
    if(valor != null){
        double resultado = valor * METRO_A_KILOMETRO;
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("Resultado de la Conversión");
        alert.setHeaderText("Conversion Exitosa");
        alert.setContentText(
            String.format("El resultado de la conversion de Metros a Kilometros es:
%.2f",
                resultado));
        alert.showAndWait();
    }else{
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle("Ventana de Error");
        alert.setHeaderText("Error");
        alert.setContentText("Debe ingresar un numero");
        alert.showAndWait();
    }
}
```



Ahora faltan los otros tres botones. Sin embargo, podemos notar que todos los botones van a realizar lo mismo, pero con diferentes valores, entonces vamos a extraer las funciones para poder reutilizarlas.

Primero la más sencilla, la de mostrar la alerta de error. La podemos mover a una función aparte de esta forma

```
public void mostrarError(){  
    Alert alert = new Alert(AlertType.ERROR);  
    alert.setTitle("Ventana de Error");  
    alert.setHeaderText("Error");  
    alert.setContentText("Debe ingresar un numero");  
    alert.showAndWait();  
}
```

Luego solamente la llamamos en el else así

```
else{  
    mostrarError();  
}
```

Ahora para la conversión de los datos vamos a crear una función de esta forma

```
public void mostrarResultado (String tipoConversion, double valor ){  
    Alert alert = new Alert(AlertType.INFORMATION);  
    alert.setTitle("Resultado de la conversion");  
    alert.setHeaderText("Conversion Exitosa");
```

```
String resultado = String.format("El valor de la conversion de %s es %.2f",  
    tipoConversion, valor);  
alert.setContentText(resultado);  
alert.showAndWait();  
  
}
```

La función `mostrarResultado`, recibe dos parámetros: el método, es decir el botón que fue seleccionado y el valor que acabamos de convertir.

Finalmente mostramos la ventana.

Entonces, con estas funciones, el método que se llama al hacer clic en el botón lo podemos reescribir así:

```
public void metrosKilometros(){  
    Integer valor = obtenerValor();  
    if(valor != null){  
        double resultado = valor * METRO_KILOMETRO;  
        mostrarResultado("Metros a Kilometros", resultado);  
    }else{  
        mostrarError();  
    }  
}
```

Cómo podemos reutilizar las funciones, los demás métodos quedan de la siguiente forma

```
public void metrosMillas(){
    Integer valor = obtenerValor();
    if(valor != null){
        double resultado = valor * METRO_MILLA;
        mostrarResultado("Metros a Millas", resultado);
    }else{
        mostrarError();
    }
}

public void kilometrosMillas(){
    Integer valor = obtenerValor();
    if(valor != null){
        double resultado = valor * KILOMETRO_MILLA;
        mostrarResultado("Kilometros a Millas", resultado);
    }else{
        mostrarError();
    }
}

public void centimetrosPulgadas(){
    Integer valor = obtenerValor();
    if(valor != null){
        double resultado = valor * CENTIMETRO_PULGADA;
        mostrarResultado("Centimetros a pulgadas", resultado);
    }
}
```

```
}else{  
    mostrarError();  
}  
}
```

---

## ***Descargo de responsabilidad***

La información contenida en este documento descargable en formato PDF o PPT es un reflejo del material virtual presentado en la versión online del curso. Por lo tanto, su contenido, gráficos, links de consulta, acotaciones y comentarios son responsabilidad exclusiva de su(s) respectivo(s) autor(es) por lo que su contenido no compromete al área de e-Learning del Departamento GES o al programa académico al que pertenece.

El área de e-Learning no asume ninguna responsabilidad por la actualidad, exactitud, obligaciones de derechos de autor, integridad o calidad de los contenidos proporcionados y se aclara que la utilización de este descargable se encuentra limitada de manera expresa para los propósitos educativos del curso.

