

Tema 12: Entorno Gráfico:

01 Introducción

Programación

Desarrollo de Aplicaciones Multiplataforma.

Índice de Contenidos

1.	Introducción.....	3
1.1.	Introducción	3
1.2.	Aplicación con NetBeans	3
1.2.1.	Simple JavaFX.....	3
1.2.2.	FXML JavaFX	7
1.3.	GUI.....	10
1.3.1.	Modelo - Vista - Controlador	10
1.3.2.	FXML:	10
1.3.3.	SceneBuilder:	11
1.3.4.	Generar clase controlador	12
1.3.5.	Carga de un fichero FXML.....	13
1.4.	Ejemplo Guiado – Formulario de Login	14

1. Introducción

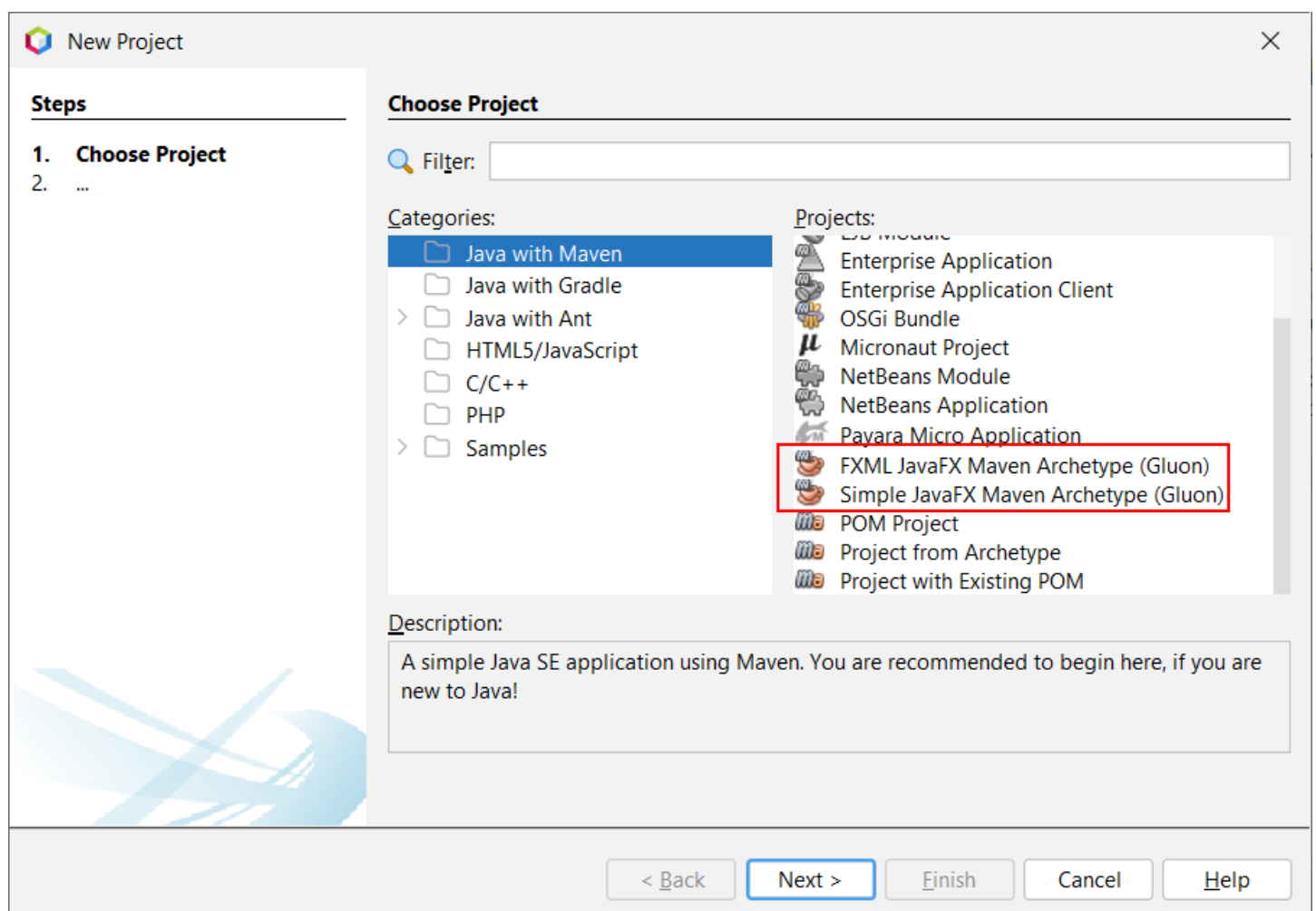
1.1. Introducción

1.2. Aplicación con NetBeans

Vamos a crear nuestra primera aplicación gráfica.

1.2.1. Simple JavaFX

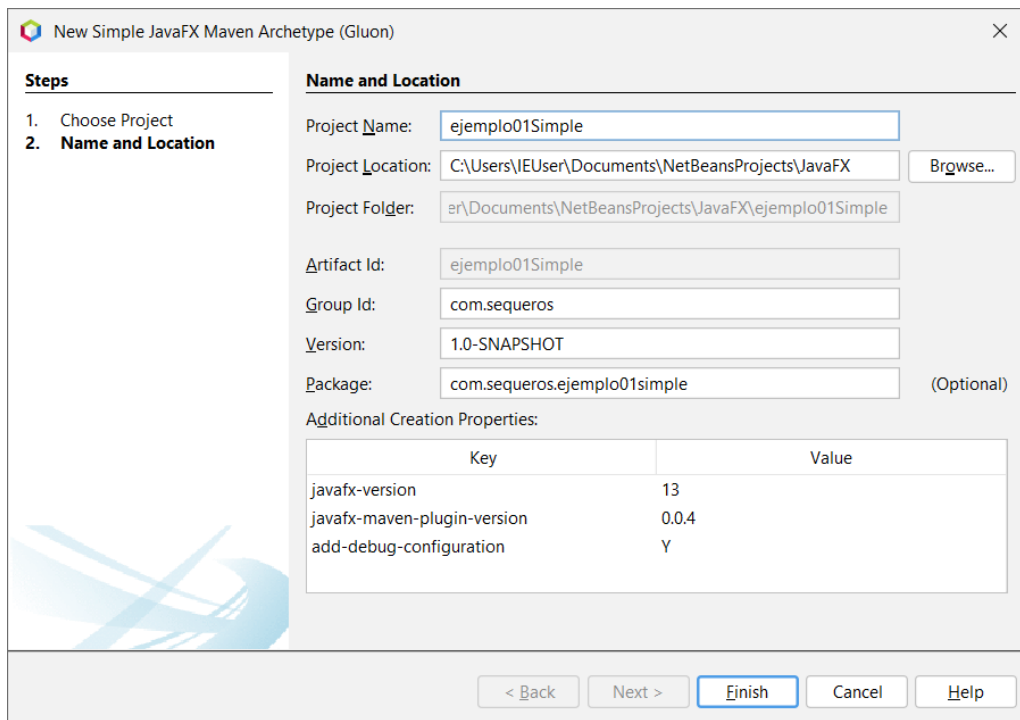
Creemos un nuevo proyecto seleccionando Java con Maven (un gestor de dependencias), nos muestra los distintos tipos de proyectos que podemos crear, hay dos con JavaFX



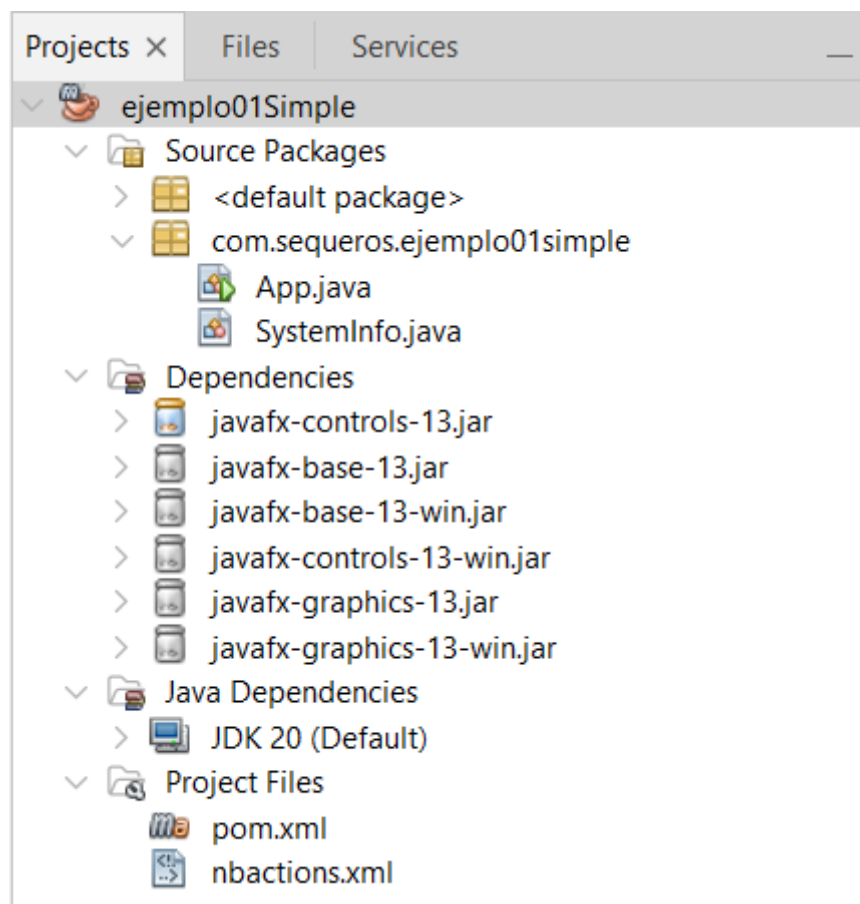
Seleccionamos el Simple y Next

Le damos nombre al proyecto, indicando la localización de este y el Group Id.

Pulsamos Finish.



Y nos crea el proyecto.

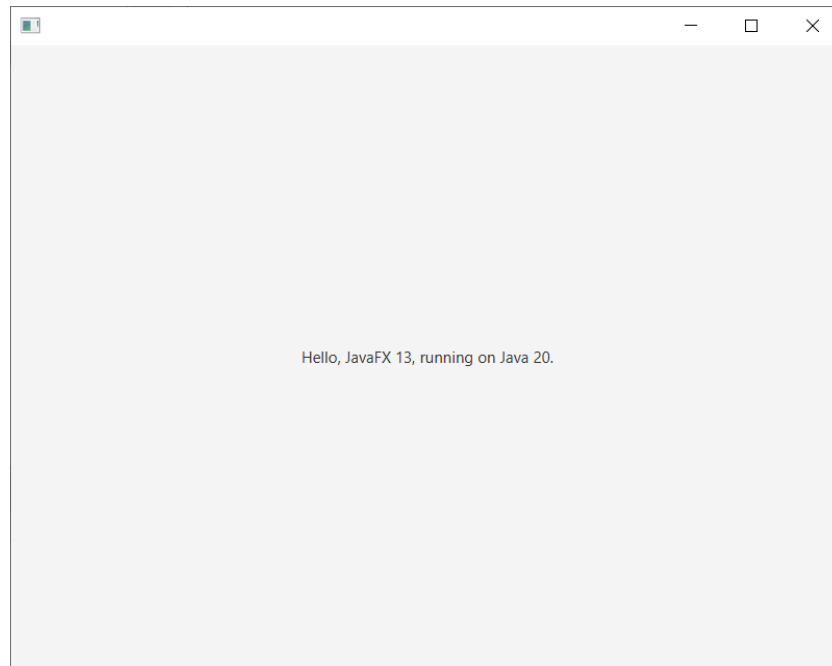


Vemos la estructura de directorios que ha creado:

```
...\JavaFX\ejemplo01Simple\src\main\java\com\sequeros\ejemplo01simple
```

Los .java de la aplicación, las dependencias utilizadas (definidas en el fichero pom.xml)

Si ejecutamos nuestro programa vemos que nos ha creado una ventana con un texto en el centro:



Este es el código:

```
1 package com.sequeros.ejemplo01simple;
2
3 import javafx.application.Application;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Label;
6 import javafx.scene.layout.StackPane;
7 import javafx.stage.Stage;
8
9
10 /**
11  * JavaFX App
12  */
13 public class App extends Application {
14
15     @Override
16     public void start(Stage stage) {
17         var javaVersion = SystemInfo.javaVersion();
18         var javafxVersion = SystemInfo.javafxVersion();
19
20         var label = new Label("Hello, JavaFX " + javafxVersion + ", running on Java " + javaVersion + ".");
21         var scene = new Scene(new StackPane(nodes: label), d: 640, dl: 480);
22         stage.setScene(scene);
23         stage.show();
24     }
25
26     public static void main(String[] args) {
27         launch();
28     }
29
30 }
```

Al ejecutar el programa vemos que tenemos una ventana y un contenido en ella. Estos componentes se identifican:

- `stage`: la ventana de la aplicación, llamada escenario
- `scene`: el contenido de la ventana, llamada escena

Vemos que hemos tenido que importar bastantes paquetes de JavaFx para que este el ejemplo funcione.

Después de importar, declaramos la clase. Tenemos que extender la clase **Application**. Con ello heredamos la funcionalidad necesaria para hacer de esta clase una aplicación basada en la interfaz gráfica de usuario.

Este programa JavaFx tiene un método principal, que realiza una llamada a un método llamado `launch`, que realizará la configuración necesaria para hacer de esta una aplicación javafx.

Después, `launch` llamará al método `start` que realiza el trabajo de crear y organizar el escenario y su vista o en otras palabras la ventana y su contenido. Observa que se pasa un parámetro de tipo **Stage** a `start`. La elección es nuestra en cuanto a cómo llamamos este parámetro, al igual que para el método `main`, podríamos llamar simplemente al array de cadenas como quisiéramos siempre que fuera un identificador válido.

En el cuerpo del método `start`, podemos hacer cosas como establecer el título de la ventana, asignar al escenario una escena, y luego mostrar el escenario o la ventana en la pantalla.

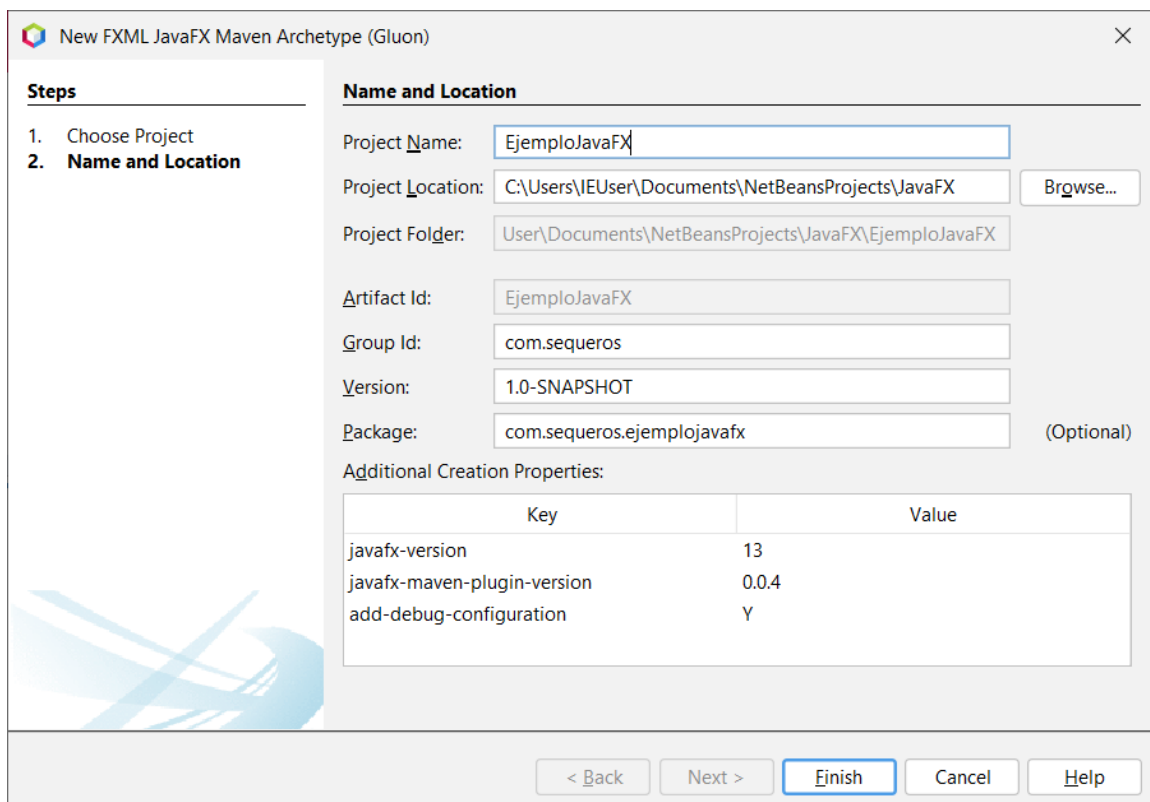
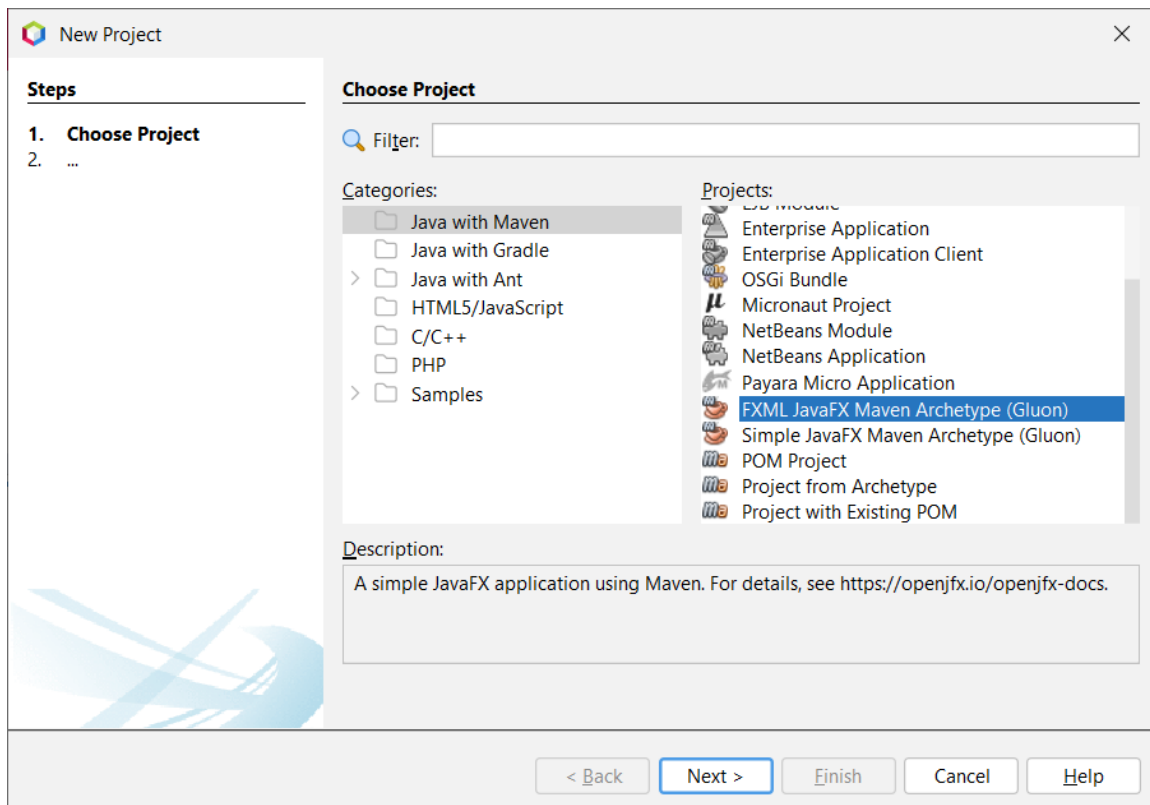
En este ejemplo se ha creado un objeto **Label** en un **StackPane** que será el nodo raíz de la escena.

La clase **stackPane** representa una especie de layout para una escena en nuestra aplicación. El layout es una especie de contenedor de elementos de la GUI. Los elementos colocados en el layout es una distribución de pila, cada elemento se coloca encima del elemento añadido anteriormente. Se puede utilizar este tipo de disposición para crear un cierto efecto de capas. En este caso, sin embargo, sólo estamos añadiendo un elemento. Así que realmente estamos usando un diseño en su forma más simple.

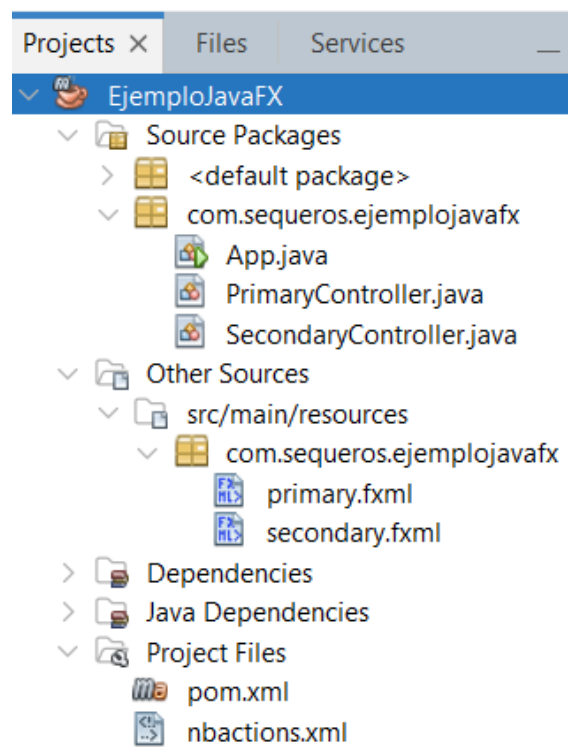
Una vez que tenemos un objeto de escena, el programa añade la escena al escenario principal y luego muestra esa escena mediante el método `show`.

1.2.2. FXML JavaFX

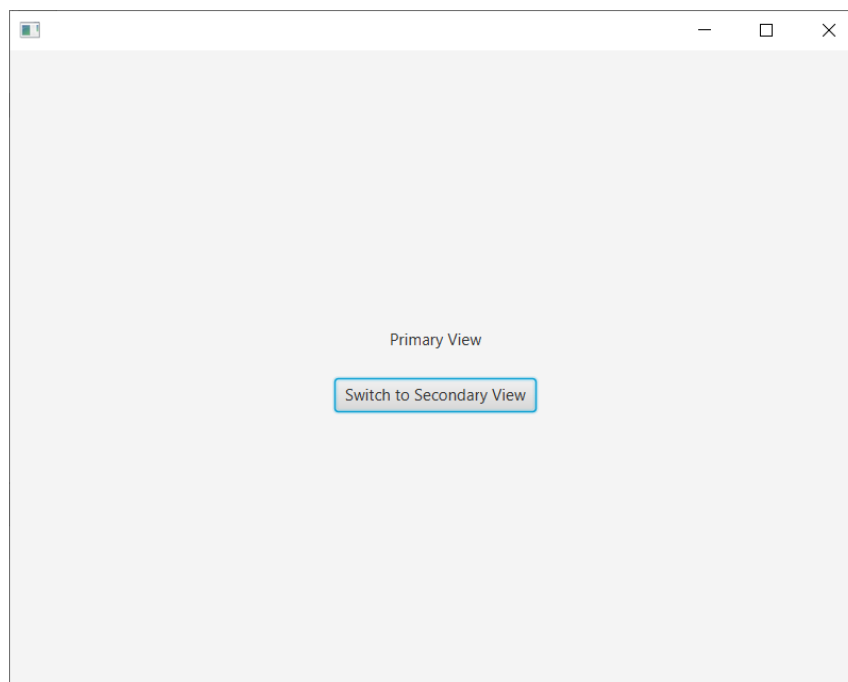
Vamos a crear otra aplicación. Seleccionamos ahora FXML JavaFX Maven Archetype:

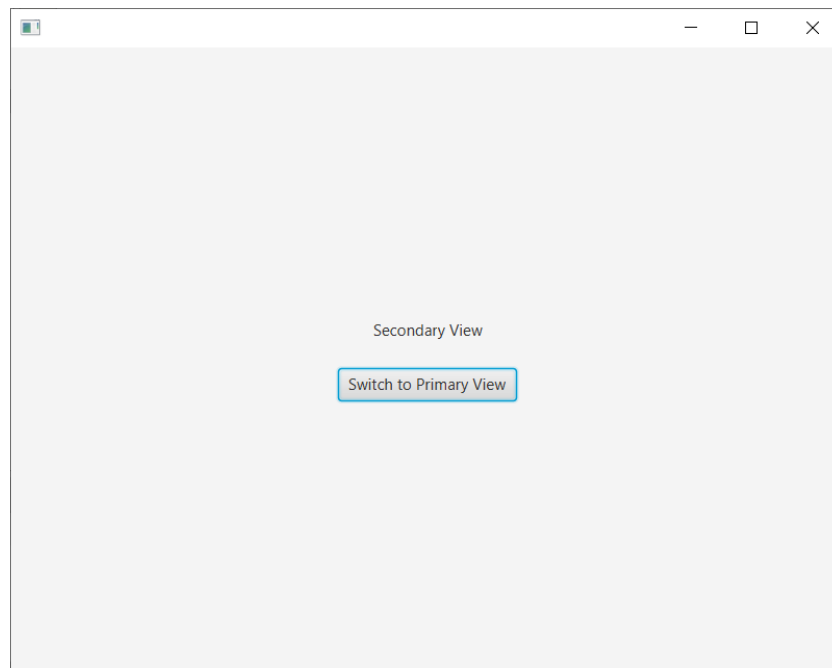


Vemos lo que se ha generado:



La ejecutamos:





Tenemos dos ventanas (Scene), conteniendo una etiqueta (Label) y un botón (Button), al hacer clic en el botón pasamos de una ventana a la otra.

Si repasamos los ficheros generados en la aplicación, vemos que tenemos:

- App.java: el main de nuestra aplicación
- primary.fxml y secondary.fxml: las dos escenas de nuestra aplicación
- PrimaryController.java y SecondaryController.java: los controladores de nuestras escenas.

1.3. GUI

1.3.1. Modelo - Vista - Controlador

Modelo-Vista-Controlador (MVC) es un patrón de diseño que separa los datos, la interfaz y la lógica de la aplicación.

- **Modelo:** no sabe nada del controlador/vista. Representa los datos (estado) y la lógica de la aplicación
- **Vista:** es la presentación visual del modelo (los datos), no puede cambiar el modelo directamente y puede ser notificada cuando hay un cambio de estado del modelo
- **Controlador:** reacciona a la petición del usuario, ejecuta la acción adecuada y actualiza el modelo pertinente, o notifica cambios en el modelo a la vista.

1.3.2. FXML:

- Los ficheros FXML contienen la descripción total o parcial del grafo de escena que representa una interfaz de usuario. El fichero tiene formato XML. En tiempo de ejecución se utiliza para crear los objetos de java de la escena (node) y componerlos en el grafo de escena. Además puede incluir los nombres de los manejadores de eventos asociados a estos objetos.
- Es similar a lo que ocurre con HTML y como se trabaja con Android, los beneficios son:
 1. Que el diseñador puede trabajar con la interfaz mientras que el programador puede trabajar con el código sin necesidad de trabajar sobre el mismo fichero
 2. Se obliga a mantener la separación entre vista y controlador

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

<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Button?>
<?import javafx.geometry.Insets?>

<VBox alignment="CENTER" spacing="20.0" xmlns="http://javafx.com/javafx/8.0.171"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.mycompany.holamundo.PrimaryController">
<children>
<Label text="Primary View" />
<Button fx:id="primaryButton" text="Switch to Secondary View" onAction="#switchToSecondary"/>
</children>
<padding>
<Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
</padding>
</VBox>
```

En JavaFX la clase Controladora es una clase Java que contiene referencias (variables) a los objetos de la interfaz y los métodos (manejadores) que se encargan de atender los eventos sobre estos objetos

Un fichero FXML debe contener una clase Controladora (normalmente llamada Controller) que actuará como controlador de la interfaz definida en el fichero. Un objeto de esta clase es creado cuando se carga el fichero

Para enlazar las variables definidas en el controlador con los objetos creados en tiempo de ejecución se utiliza inyección. Consiste en asignar la referencia a los objetos definidos en la clase no en el constructor sino en el momento de creación del grafo de escena (FXMLLoader.load())

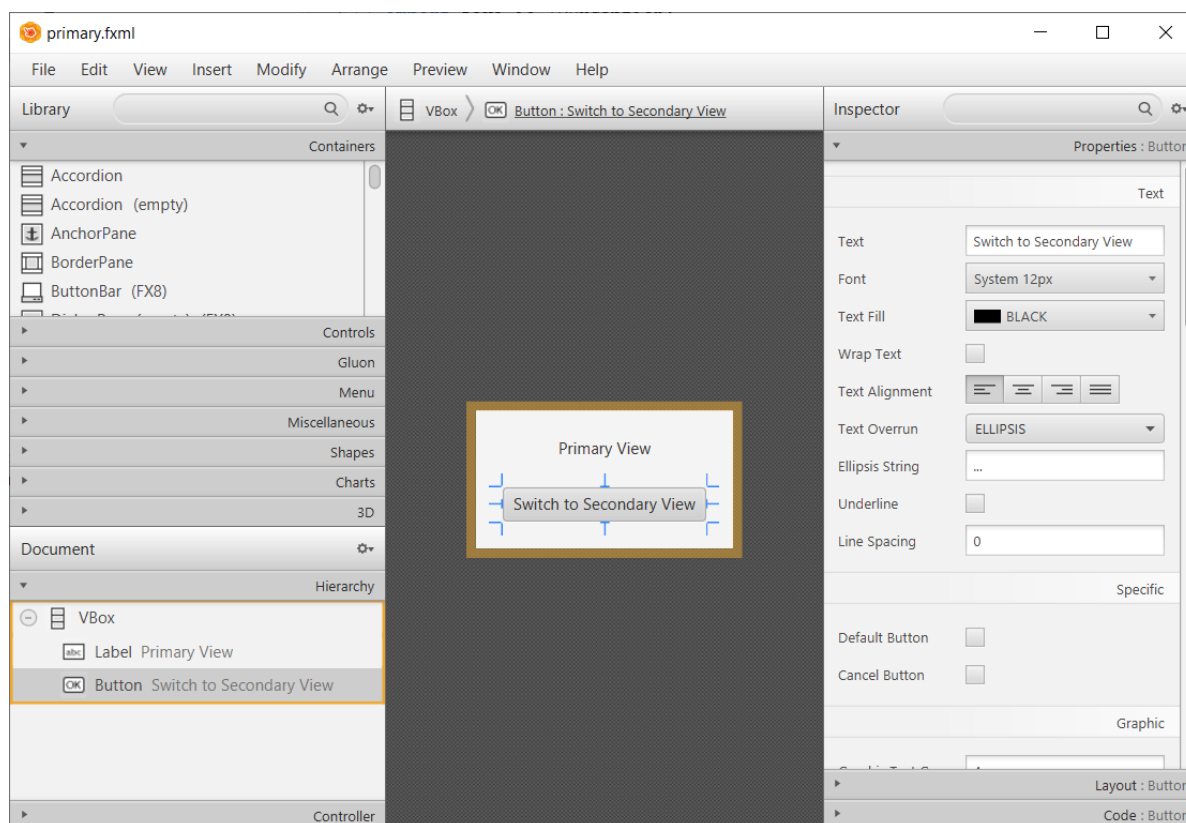
También se automatiza el registro de los manejadores sobre los objetos. De esta manera se reduce considerablemente el código necesario para crear y registrar los nodos de la escena.

1.3.3. SceneBuilder:

SceneBuilder es un editor externo de ficheros FXML desarrollado por Oracle y ahora continuado por Gluon <https://gluonhq.com/products/scene-builder/> que nos permite diseñar nuestras interfaces de forma visual

En SceneBuilder disponemos de todos los controles y contenedores definidos para JavaFX, la configuración de estos se realiza arrastrando y soltando sobre el área de trabajo dichos controles, ajustando las propiedades de los controles en un panel separado. El resultado se almacena en un fichero XML (con extensión FXML)

Se puede integrar con Netbeans, Eclipse, IntelliJ



Para añadir un elemento lo arrastraremos desde la librería de controles hasta la zona de trabajo o hasta la jerarquía de controles en el documento. • Es posible filtrar los controles por nombre • Con el control seleccionado podremos modificar cualquiera de sus propiedades. Las propiedades son los atributos disponibles de cada control (posición, tamaño, apariencia, etc.)

En el panel Inspector tenemos las secciones Properties, Layout y Code:

- La sección Properties nos permite definir el estilo del elemento seleccionado en el área de trabajo. En JavaFX se utiliza plantillas CSS para definir el estilo de los elementos (lo veremos más adelante)
- La sección Layouts nos permite especificar el comportamiento en tiempo de ejecución del contenedor cuando cambiamos el tamaño de la ventana. También permite definir el tamaño del control. La información que aparece en esta sección dependerá del contenedor

- La sección Code especifica los métodos que se ejecutarán ante la interacción del usuario sobre el control. El campo fx:id determina el nombre de la variable que tendrá este objeto dentro de la clase controladora.
 - Esta sección es muy importante para relacionar correctamente el diseño con el código Java. Además de definir el nombre del objeto podemos asignar los manejadores de eventos. Para aprovecharla generación automática de código desde NetBeans, es recomendable dar nombre a los manejadores en el SceneBuilder

Para asignar una clase Java Controlador debemos de añadirlo en el panel situado la parte inferior de la jerarquía del documento.

Con el comando “Wrap in” situamos los controles seleccionados dentro de uno de los contenedores disponibles

El comando “Unwrap” elimina el contenedor seleccionado pero deja sus controles inalterados

Con el comando “Fit to Parent” cambiamos el tamaño del control seleccionado hasta que ocupe el área de su contenedor

El comando “Use Computed Sizes” resetea los valores de las propiedades del contenedor a USE_COMPUTED_SIZE

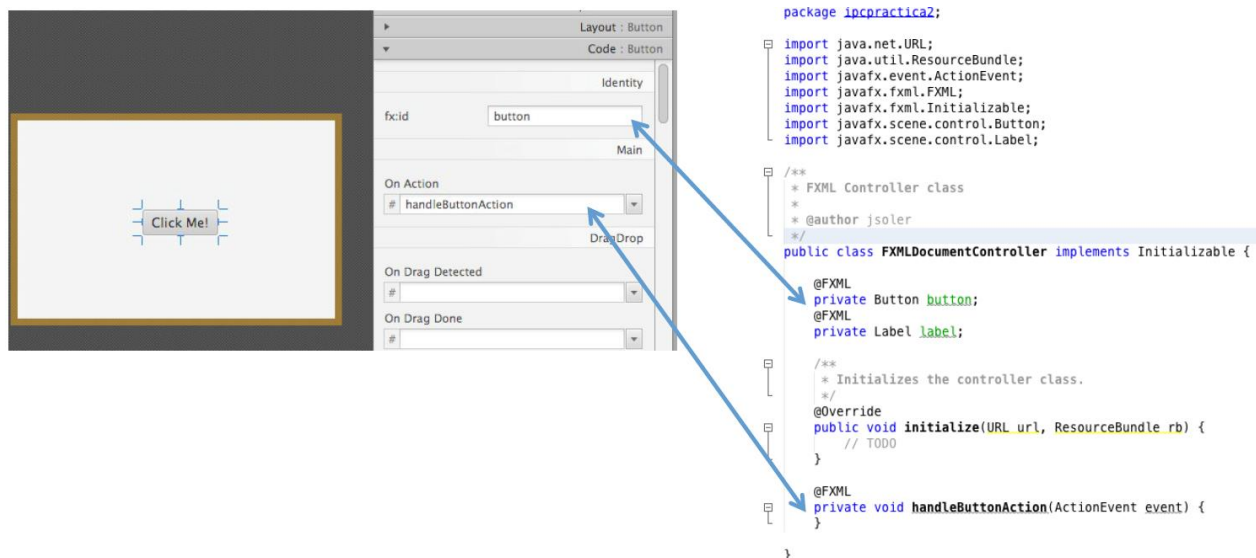
El comando “Show/Hide Simple Data” muestra datos ficticios en aquellos controles del tipo lista, tabla o árbol. Los datos no se guardan en el fichero FXML

El comando “Show Preview” muestra en una ventana el resultado final del fichero FXML que se está editando

El comando “Show Sample Controller Skeleton” abre una ventana y muestra una plantilla de código para crear una clase controlador a partir del fichero FXML

1.3.4. Generar clase controlador

- Desde el explorador de NetBeans seleccionaremos el fichero FXML y con el botón derecho accederemos al menú Make Controller • Si el fichero de la clase Controlador ya existe, éste se actualizara con los datos del fichero XML. En ningún caso borra el código existente.
- Además de generar la Clase Controlador con los manejadores y las referencias a los controles, modifica el fichero FXML y le añade la referencia a la clase controlador.



1.3.5. Carga de un fichero FXML

Un fichero FXML se carga utilizando el método `load()` de la clase `FXMLLoader`.

Hay dos opciones, utilizar el método estático de la clase o crear previamente una instancia de la clase:

- (opción 1)

```
Parent raiz= FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));
```

- (opción 2)

```
FXMLLoader loader= new FXMLLoader(getClass().getResource("FXMLDocument.fxml"));  
Parent raiz = loader.load();
```

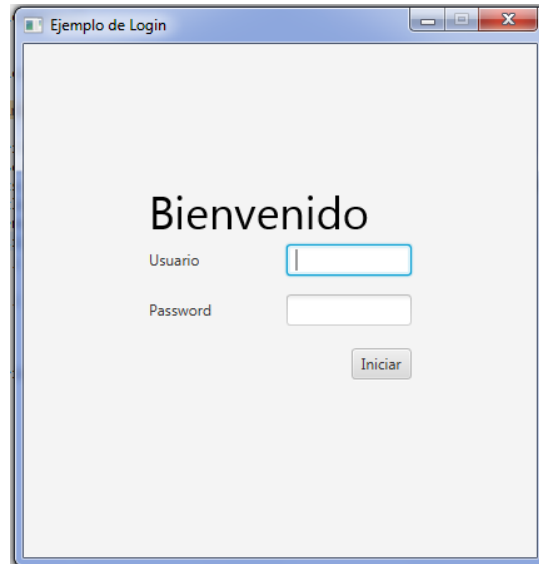
Durante la carga del fichero se realizan las siguientes tareas:

- Se crea un objeto de la clase controladora
- Se crean los objetos definidos en el fichero FXML, se les asignan manejadores y se construye el grafo de escena
- Mediante inyección se enlazan las variables de la clase controladora con los objetos creados en la etapa anterior.
- Se ejecuta el método `Initialize` en el objeto de la clase controladora (si está definido). Es en este método donde añadiremos la inicialización adicional que necesite nuestra aplicación.

1.4. Ejemplo Guiado – Formulario de Login

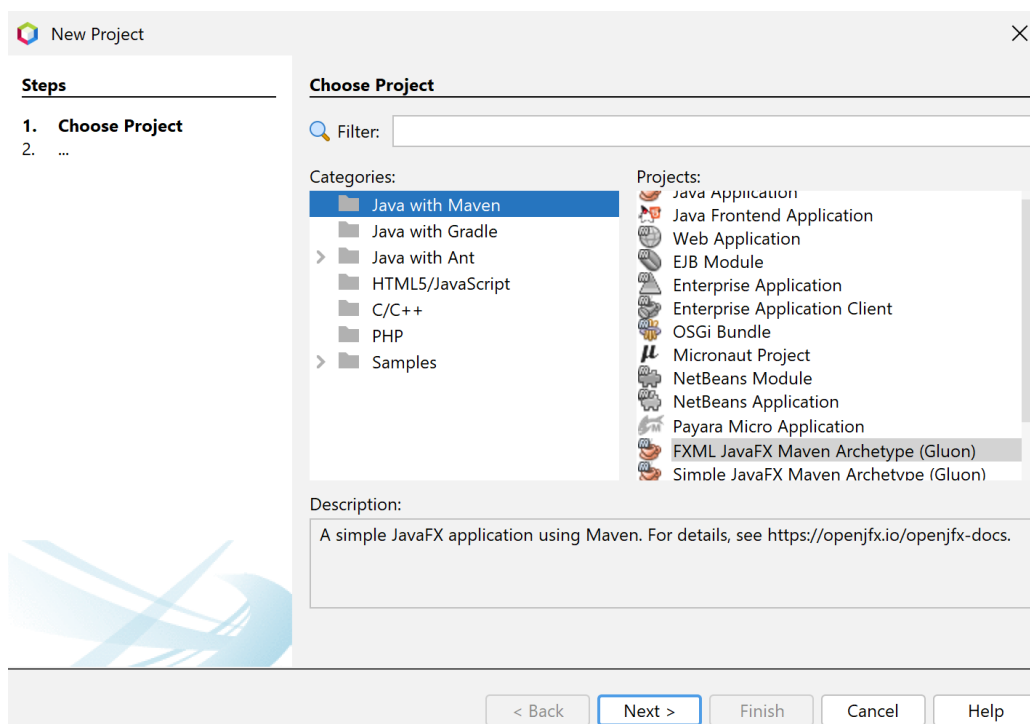
El objetivo del ejercicio es diseñar con Scene Builder una interfaz de usuario que simule el login de un usuario en una aplicación. Utilizaremos un manejador de eventos para mostrar en pantalla un texto de bienvenida cada vez que se pulsa el botón Iniciar del formulario.

La apariencia final del formulario es la que se muestra en la Figura:



1. Crear el proyecto

Dentro del entorno NetBeans utilice File -> NewProject y seleccione un proyecto Java with Maven - FXML JavaFX Maven Archetype (Gluon)



Pulse el botón Next y en la siguiente pantalla ponga un nombre al proyecto, EjercicioLogin y después pulse Finish:

New FXML 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:

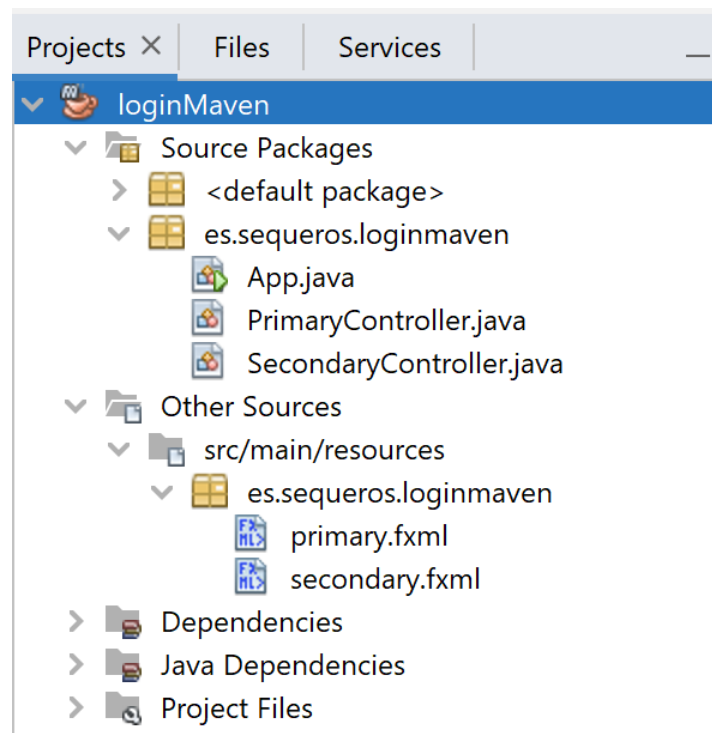
Package: (Optional)

Additional Creation Properties:

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

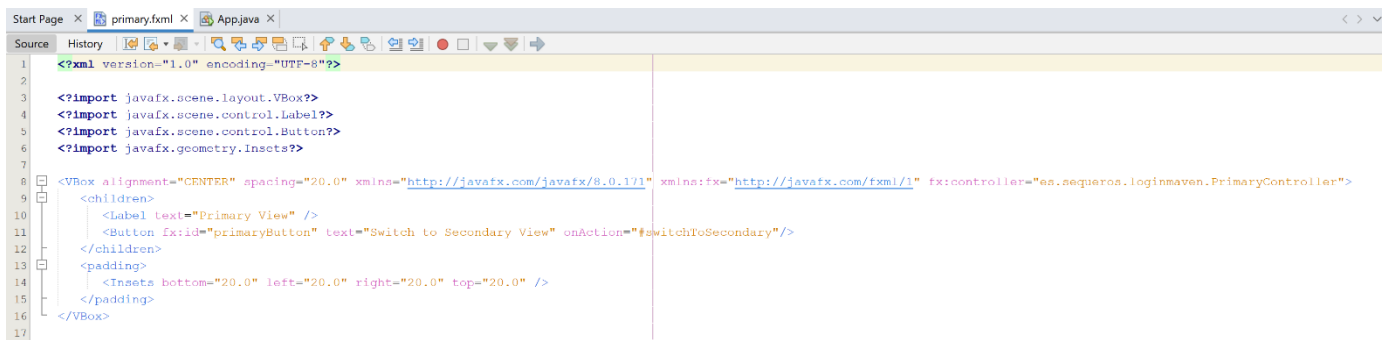
< Back Next > **Finish** Cancel Help

NetBeans ha creado la estructura del proyecto, tal como se muestra en la Figura



Por defecto Netbeans ha creado:

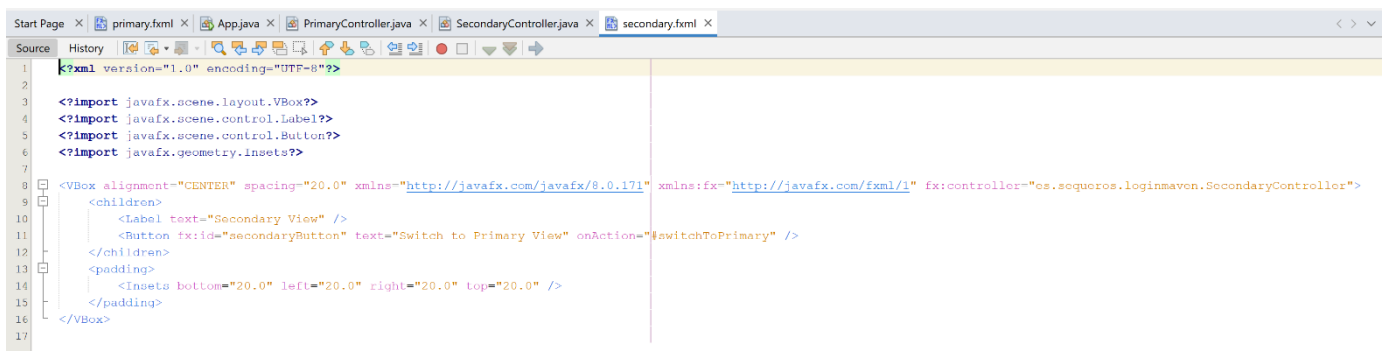
- Dos ficheros XML, con el siguiente contenido:



```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.layout.VBox?>
4 <?import javafx.scene.control.Label?>
5 <?import javafx.scene.control.Button?>
6 <?import javafx.geometry.Insets?>
7
8 <VBox alignment="CENTER" spacing="20.0" xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller="es.sequeros.loginmaven.PrimaryController">
9   <children>
10    <Label text="Primary View" />
11    <Button fx:id="primaryButton" text="Switch to Secondary View" onAction="#switchToSecondary"/>
12  </children>
13  <padding>
14    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
15  </padding>
16 </VBox>
17

```

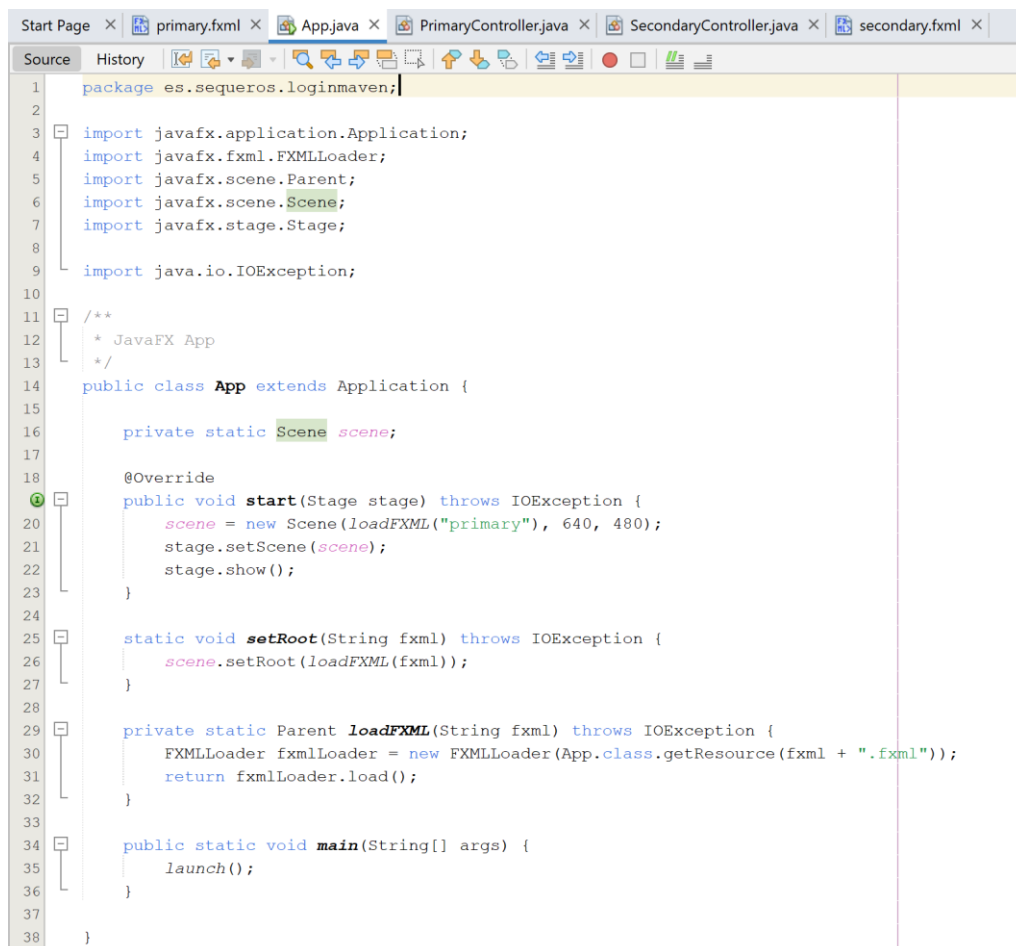


```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.layout.VBox?>
4 <?import javafx.scene.control.Label?>
5 <?import javafx.scene.control.Button?>
6 <?import javafx.geometry.Insets?>
7
8 <VBox alignment="CENTER" spacing="20.0" xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller="es.sequeros.loginmaven.SecondaryController">
9   <children>
10    <Label text="Secondary View" />
11    <Button fx:id="secondaryButton" text="Switch to Primary View" onAction="#switchToPrimary"/>
12  </children>
13  <padding>
14    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
15  </padding>
16 </VBox>
17

```

- La clase App, con el main de la aplicación:

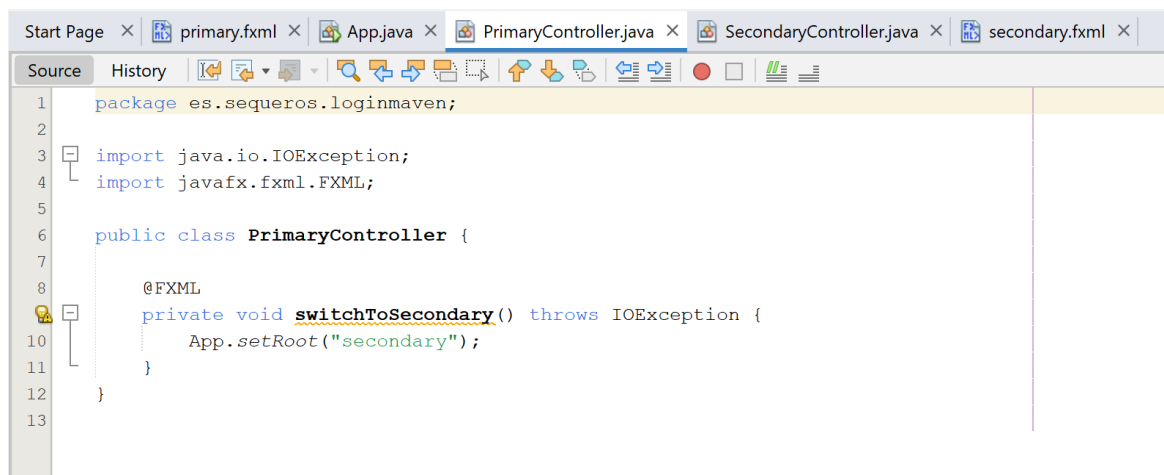


```

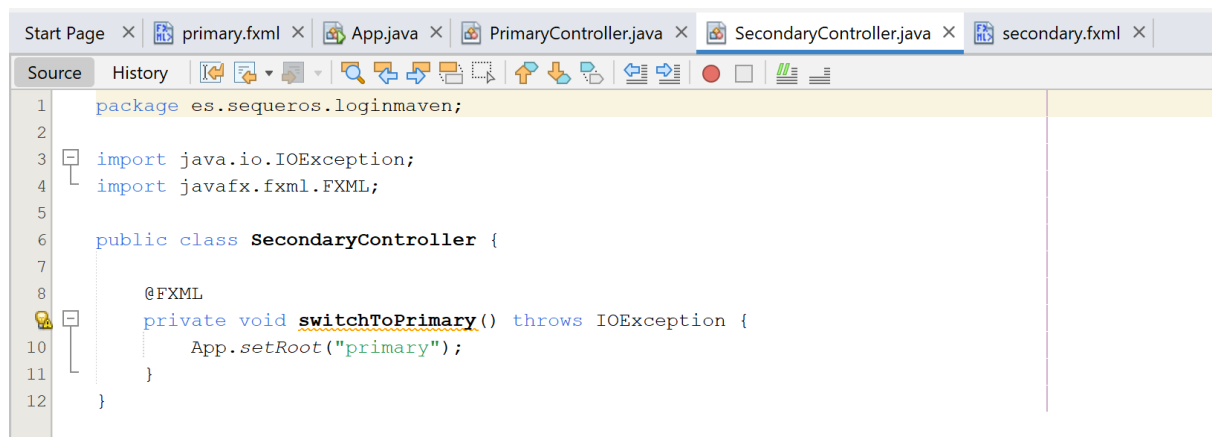
1 package es.sequeros.loginmaven;
2
3 import javafx.application.Application;
4 import javafx.fxml.FXMLLoader;
5 import javafx.scene.Parent;
6 import javafx.scene.Scene;
7 import javafx.stage.Stage;
8
9 import java.io.IOException;
10
11 /**
12  * JavaFX App
13  */
14 public class App extends Application {
15
16     private static Scene scene;
17
18     @Override
19     public void start(Stage stage) throws IOException {
20         scene = new Scene(loadFXML("primary"), 640, 480);
21         stage.setScene(scene);
22         stage.show();
23     }
24
25     static void setRoot(String fxml) throws IOException {
26         scene.setRoot(loadFXML(fxml));
27     }
28
29     private static Parent loadFXML(String fxml) throws IOException {
30         FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
31         return fxmlLoader.load();
32     }
33
34     public static void main(String[] args) {
35         launch();
36     }
37
38 }

```


- Y dos clases como controller:



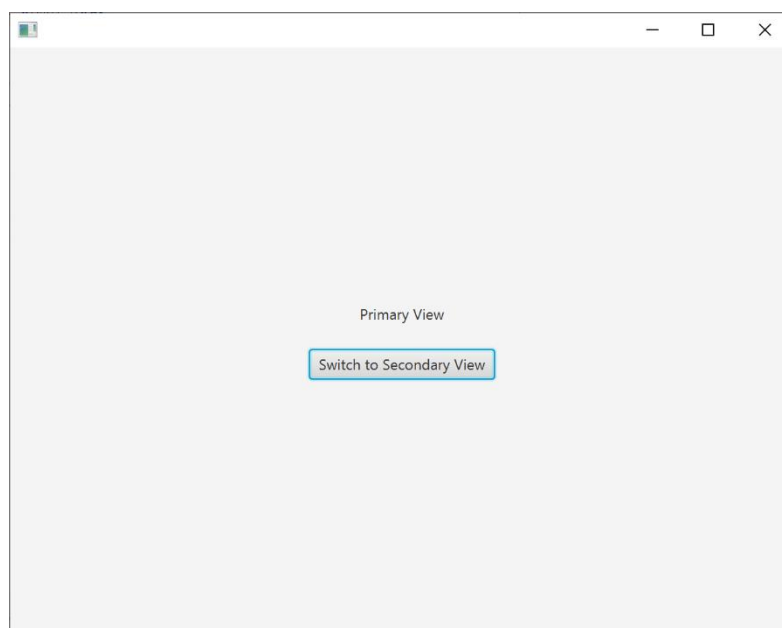
```
1 package es.sequeros.loginmaven;
2
3 import java.io.IOException;
4 import javafx.fxml.FXML;
5
6 public class PrimaryController {
7
8     @FXML
9     private void switchToSecondary() throws IOException {
10         App.setRoot("secondary");
11     }
12 }
13
```



```
1 package es.sequeros.loginmaven;
2
3 import java.io.IOException;
4 import javafx.fxml.FXML;
5
6 public class SecondaryController {
7
8     @FXML
9     private void switchToPrimary() throws IOException {
10         App.setRoot("primary");
11     }
12 }

```

Podemos ejecutar el proyecto:

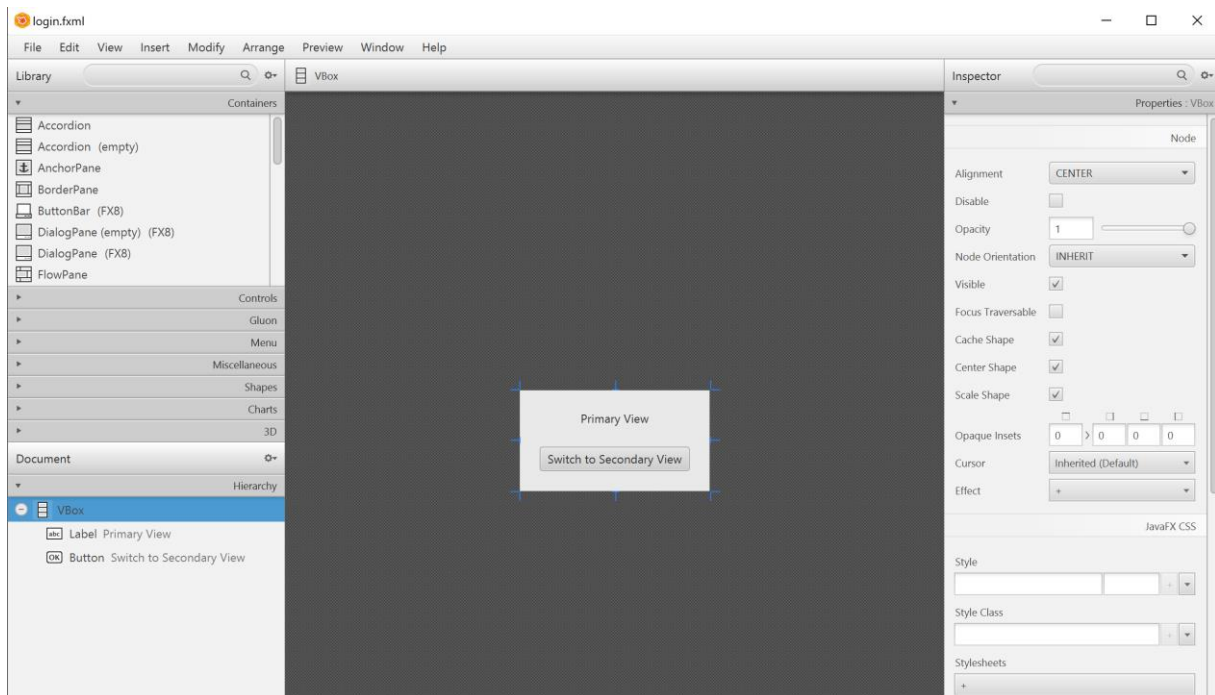


2. Crear la interfaz con Scene Builder

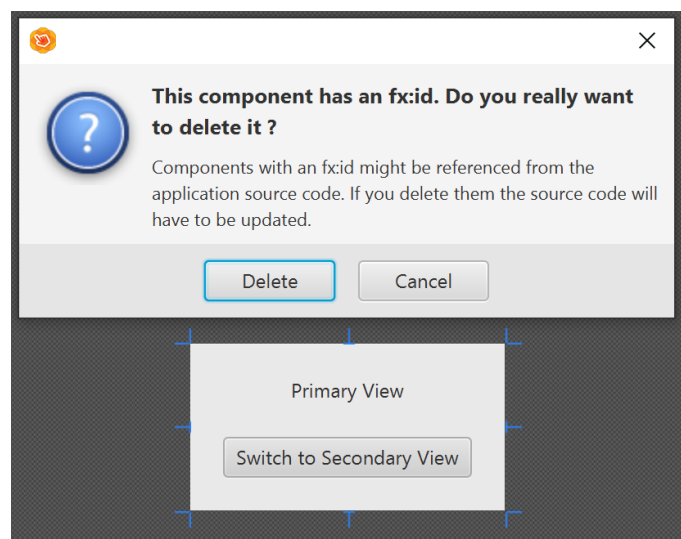
Para crear la interfaz vamos a reutilizar el fichero FXML generado por defecto.

- Renombramos el fichero primary.fxml a login.fxml
- El fichero primaryController.java no nos va a resultar útil, antes de seguir lo eliminaremos del explorador del proyecto. (no es necesario?)
- Ahora diseñaremos la interfaz de usuario, para ello marcamos el archivo login.fxml y con el botón derecho del ratón en el menú contextual Open (ó simplemente doble clic)

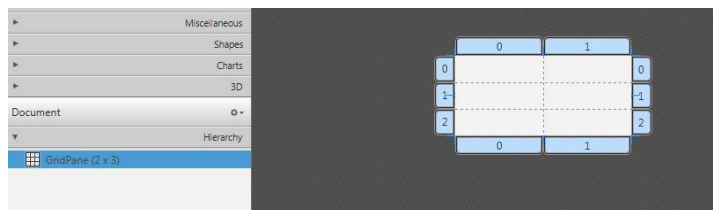
Interfaz de usuario de Scene Builder:



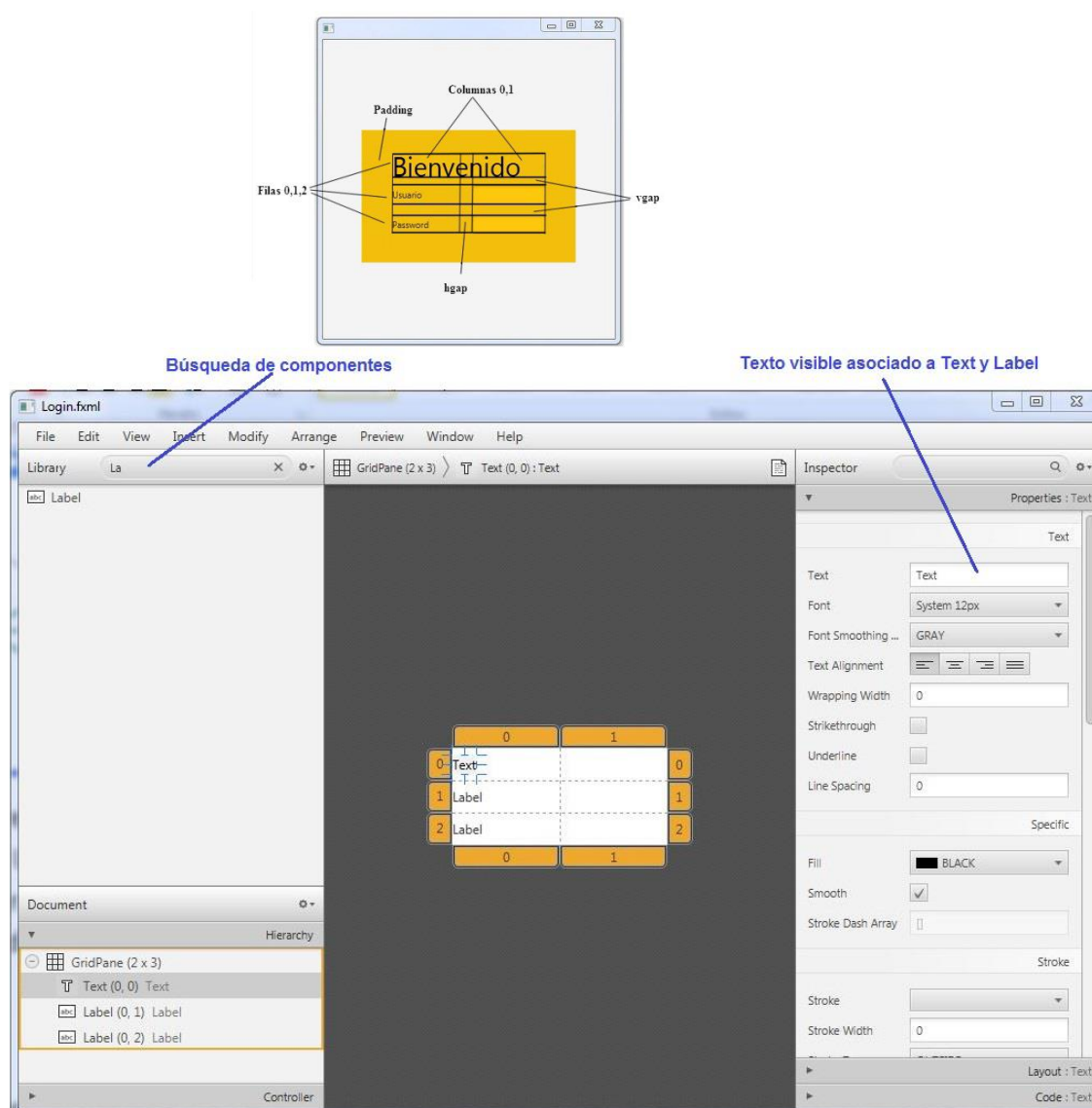
- Ahora vamos a eliminar todos los elementos gráficos del fichero (AnchorPane, Button, Label) y empezaremos a diseñar la interfaz de login. Para ello, en el panel de la izquierda, dentro de la sección Document, selecciona el AnchorPane actual y presiona Suprimir.



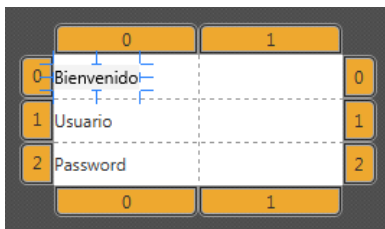
- e. Vamos a empezar añadiendo un layout GridPane que representa una matriz de filas y columnas en las cuales se pueden situar componentes de la interfaz de usuario. Definiremos una matriz de 3 filas y 2 columnas (2x3). Para ello selecciona un GridPane de la librería de controles y arrástralo a la zona de trabajo. Por defecto se creará un grid de 2x3.



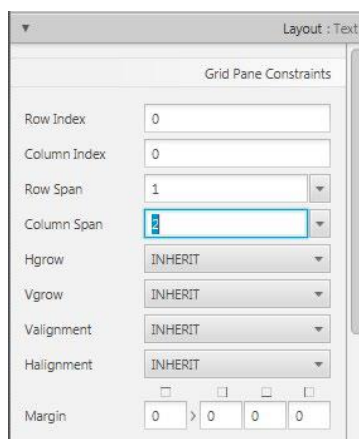
- f. Si necesitamos más filas o columnas, sobre el panel izquierdo Hierarchy y usando el menú contextual (el botón derecho del ratón) nos aparecerán todas las opciones necesarias. De la misma manera, en la zona de trabajo podemos seleccionar una fila o columna y con el botón derecho del ratón nos aparecerán las opciones para modificar el GridPane.
- g. El objetivo ahora es diseñar la ventana mostrada en la Figura:



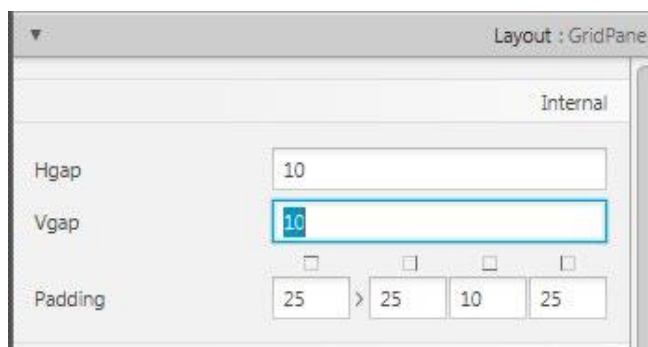
- h. Empezaremos añadiendo desde Scene Builder algunos de los componentes de la interfaz de usuario. Usaremos un componente Text para el mensaje de bienvenida y dos componentes Label para las etiquetas usuario y password. Los seleccionamos en la paleta (puede usar la función de búsqueda en Library, tal y como se indica en la Figura) y los dejamos respectivamente en las tres filas de la matriz.
- i. Escriba en la propiedad Text de cada uno de los componentes: Bienvenido, Usuario y Password.



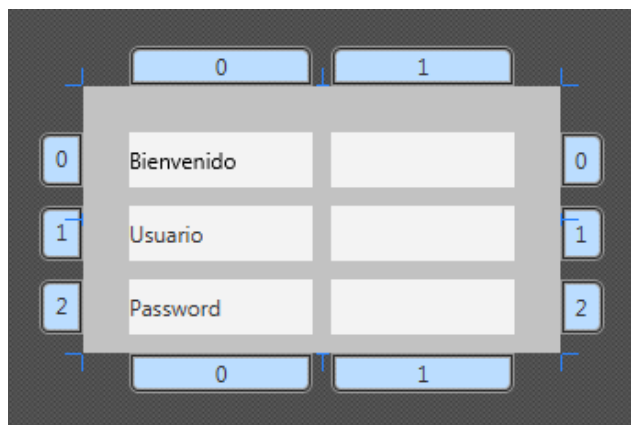
- j. Ahora cambiaremos algunas propiedades del componente Text (Bienvenido). Abra dentro del inspector la pestaña Layout y escriba en Columnspan 2 (Figura 13). Esto permite que el componente pueda ocupar, si es necesario, la segunda columna.



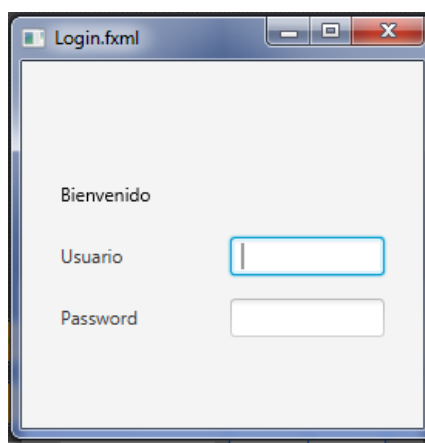
- k. Ahora definiremos la separación entre filas (Vgap) y entre columnas (Hgap). Seleccione el GridPane y en Layout teclee los valores 10, 10 en Vgap y Hgap. Fijaremos también el margen interno (padding) del gridPane. En Padding introduzca 25,25,10,25.



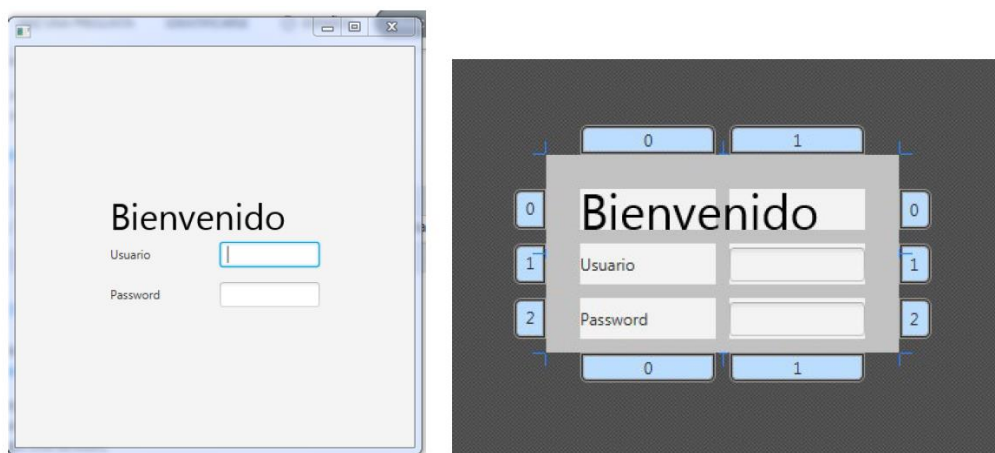
- l. El aspecto del componente es el de la Figura 14.



- m. Añadiremos ahora un componente para la entrada de texto (TextField) y otro para la entrada de la contraseña (PasswordField). Los arrastramos a sus respectivas posiciones. En Preview puede ver el aspecto de la interfaz:



- n. Vamos a cambiar el tamaño de la fuente para el mensaje Bienvenido, de modo que pueda ocupar las dos columnas. Seleccione el componente Text y cambie en Properties -> Font el valor a 36.

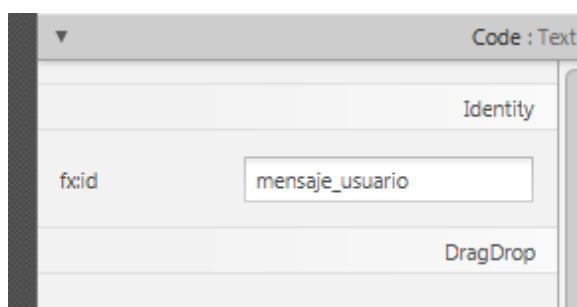


- o. Lo último que queda por añadir es un botón y el texto que muestra el inicio de sesión. Añada una nueva fila al final de la matriz y sitúe dentro en la posición 1,3 un panel HBox y dentro de éste un botón. Ponga el alineamiento del panel a Bottom-Right y cambie el texto del botón a Iniciar. Para el mensaje que recibirá el usuario utilizaremos en componente Text que se ubicará en una nueva fila de la matriz. Sitúe el componente Text en la primera columna y última fila, cambie la propiedad Fill a color rojo, use pare eso la paleta de colores. Además, cambie la propiedad Text del componente Text al string vacío.

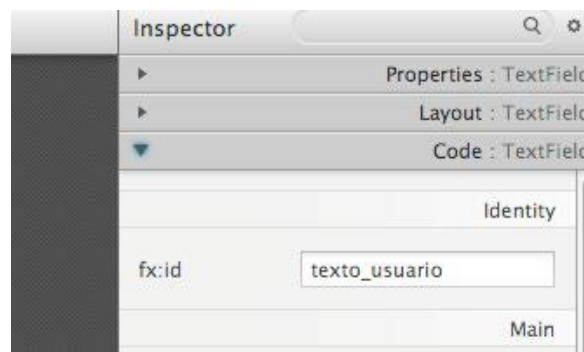


3. Acciones necesarias para añadir el comportamiento

- a. Para que el texto cambie cuando se ejecuta el programa y después de pulsar el botón Iniciar, el componente Text debe tener un ID, que luego será referenciado desde la clase controladora de la interfaz de usuario. Para ello seleccione el componente y en la pestaña Code escriba en el campo fx:id, debajo de Identity, mensaje_usuario.



- b. Vamos a hacer lo mismo en el TextField en el que se introduce el usuario, pondremos en el campo fx:id texto_usuario



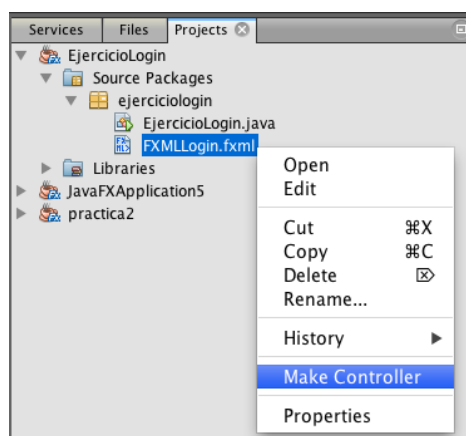
- c. Para finalizar añadiremos funcionalidad al botón, definiendo un manejador de evento que se ejecutará cuando éste resulte pulsado. Seleccione el botón Iniciar y abra la pestaña del Inspector Code y busque On Action, incluya allí un nombre descriptivo como por ejemplo pulsadoIniciar. Esto será el nombre de método que posteriormente tiene que ser implementado en la clase controlador.



- d. Antes de abandonar SceneBuilder es necesario salvar el fichero con el que estamos trabajando.

4. Crear la clase de Java que se asocia como controlador

- a. El código de manejo del evento lo situamos en la clase controladora que vamos a crear de manera automática en NetBeans. Para ello seleccionaremos en el explorador del proyecto el fichero FXMLLogin y con el botón derecho de ratón invocaremos a Make Controller



- b. Se crea automáticamente el fichero FXMMLLoginController, si ya existe se actualiza con los cambios introducidos en el fichero FXML. En la Figura 22 se muestra el resultado.

```
package es.sequeros.loginmaven;

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

import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.text.Text;

/**
 * FXML Controller class
 *
 * @author IEUser
 */
public class LoginController implements Initializable {

    @FXML
    private TextField texto_usuario;
    @FXML
    private PasswordField password_usuario;
    @FXML
    private Text mensaje_usuario;

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    @FXML
    private void pulsadoIniciar(ActionEvent event) {
    }

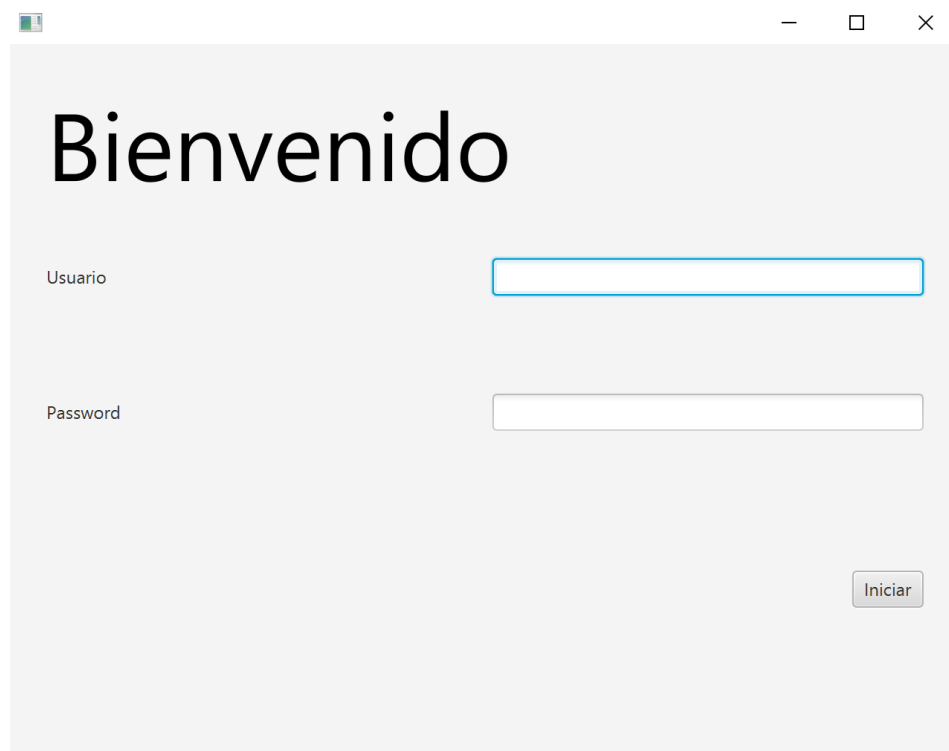
}
```

Como veis aparecen debajo de las anotaciones @FXML los elementos que en el fichero FXML tienen asignado un valor en el campo fx:id así como el método que hemos indicado en el campo onAction del botón.

- c. Para que la aplicación reaccione al clic del botón tendremos que añadir la siguiente línea de código que aparece en la siguiente figura dentro del método pulsadoIniciar:

```
35
36
37 @FXML
38 private void pulsadoIniciar(ActionEvent event) {
39     mensaje_usuario.setText("Bienvenido " + texto_usuario.getText());
40 }
```


- d. Ahora podemos ejecutar la aplicación desde NetBeans mediante Run Project



- e. Si queremos que la ventana tenga un título tenemos que añadir el siguiente código `stage.setTitle("Login")` en la clase `Main.java`. La instrucción `stage.setResizable(false)` impide el redimensionamiento del formulario.

```
20  @Override
21  public void start(Stage stage) throws Exception {
22      Parent root = FXMLLoader.load(getClass().getResource("FXMLLogin.fxml"));
23
24      Scene scene = new Scene(root);
25
26      stage.setScene(scene);
27
28      stage.setTitle("Login");
29      stage.setResizable(false);
30
31      stage.show();
32  }
```