

Modelo Vista Controlador

El patrón MVC

- **Un modelo:** Es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.
- **Varias vistas:** Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida. En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generará la salida, tal como nuestra aplicación requiera.
- **Varios controladores:** Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc. En realidad es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación.
- Las vistas y los controladores suelen estar muy relacionados
 - Los controladores tratan los eventos que se producen en la interfaz gráfica (vista)

Esta separación de aspectos de una aplicación da mucha flexibilidad al desarrollador

Flujo de control:

1. El usuario realiza una acción en la interfaz
2. El controlador trata el evento de entrada
3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo
4. Se genera una nueva vista. La vista toma los datos del modelo
5. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo

MVC en Java Swing

- Modelo:
 - El modelo lo realiza el desarrollador
- Vista:
 - Conjunto de objetos de clases que heredan de `java.awt.Component`
- Controlador:
 - El controlador es el thread de tratamiento de eventos, que captura y propaga los eventos a la vista y al modelo
 - Clases de tratamiento de los eventos (a veces como clases anónimas) que implementan interfaces de tipo `EventListener` (`ActionListener`, `MouseListener`, `WindowListener`, etc.)

Vamos a realizar una aplicación y de paso ver algunos controles mediante Swing.

La pantalla de ejemplo:

Cálculo Peso Ideal

Nombre: ☒ Hombre ☐ Mujer

Apellidos:

Edad: Altura: Peso:

Nombre completo: María Álvarez
Edad: 25 Altura: 159 Peso: 55.0
IMC: 21.75546853368142
Peso Ideal: 54.5

Nombre completo: Pedro Martín
Edad: 29 Altura: 182 Peso: 85.0
IMC: 25.661152034778407
Peso Ideal: 74.0

En la aplicación realizada, el código auto-generado (en gris) sería lo más parecido a hablar de la vista.

El código que establecemos en los métodos que gestionan los eventos serían el controlador (modifican la vista introduciendo la información del usuario y del modelo. Se comunican con el modelo para obtener los resultados)

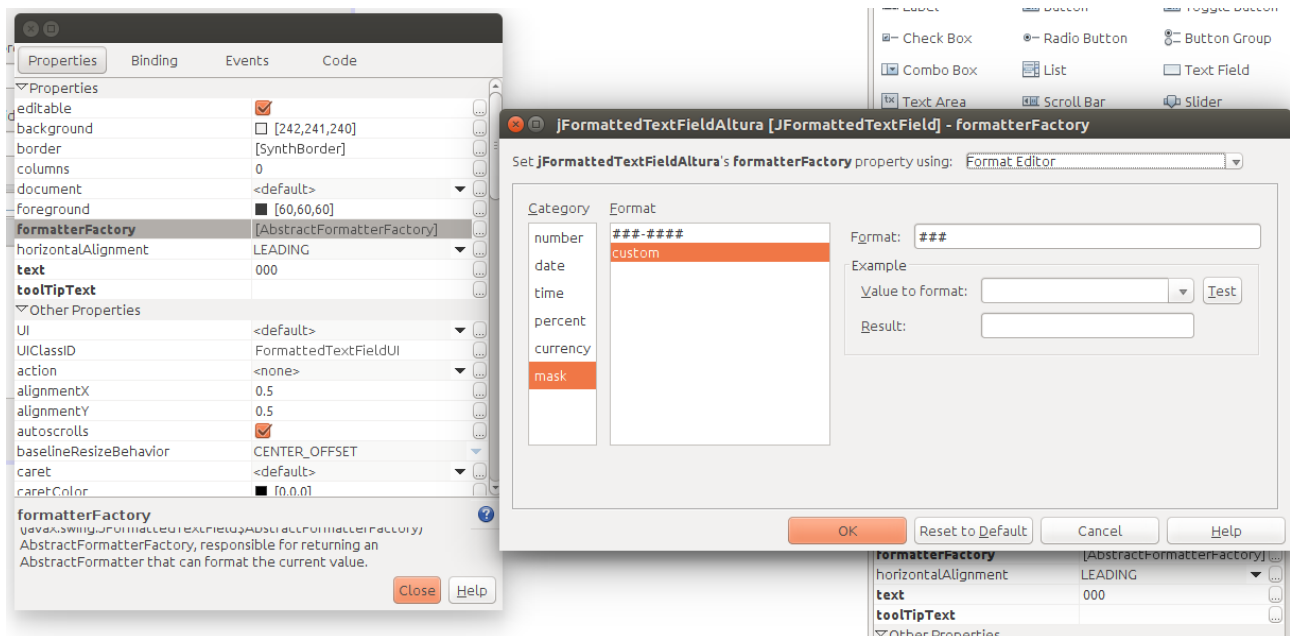
El modelo estaría compuesto por los ficheros Persona, Hombre, Mujer.

Para poner el nombre en la ventana vamos a title en el JFrame y ponemos: “Cálculo Peso Ideal”

Para conseguir que cuando marquemos una opción Hombre quite la selección en Mujer y viceversa incorporamos un ButtonGroup. Luego agregamos los dos RadioButton a ese buttongroup

Hay 3 JFormattedTextField uno para edad otro para altura y otro para peso

Veamos un ejemplo de como poner la máscara:



En las propiedades del objeto vamos al apartado FormattedFactory y pulsamos en el símbolo tres puntos. Allí seleccionamos mask --> custom

###

hace referencia a que deben ponerse 3 dígitos de otra forma no acepta ninguna otra entradas

Para establecer así como nosotros queremos el String resultante para determinar una cantidad específica de decimales y demás:

```
StringBuilder sbuf = new StringBuilder();
Formatter fmt = new Formatter(sbuf);
fmt.format("PI = %.3f%n", Math.PI);
System.out.println(sbuf);
```

Con lo anterior se consigue especificar 3 decimales al mostrar PI mediante una String

Práctica 2: Realizar la aplicación descrita mediante Netbeans con Swing

No vamos a detenernos más en Swing. Vamos a trabajar en JavaFX que es más moderno y nos permite ver todavía mejor la separación en el patrón MVC.

JavaFX

JavaFX proporciona a los desarrolladores de Java una nueva plataforma gráfica. JavaFX 2.0 se publicó en octubre del 2011 con la intención de reemplazar a Swing en la creación de nuevos interfaces gráficos de usuario (IGU).

Para realizar una aplicación en JavaFX utilizaremos el paquete netbeans que nos da oracle para descargar ya con todo incluido.

Te podría interesar mantener los siguientes enlaces:

- [Java 8 API](#) - Documentación (JavaDoc) de las clases estándar de Java
- [JavaFX 8 API](#) - Documentación de las clases JavaFX
- [ControlsFX API](#) - Documentación para el proyecto [ControlsFX](#), el cual ofrece controles JavaFX adicionales
- [Oracle's JavaFX Tutorials](#) - Tutoriales oficiales de Oracle sobre JavaFX

JavaFX es un conjunto de gráficos y paquetes de comunicación que permite a los desarrolladores para diseñar, crear, probar, depurar y desplegar aplicaciones cliente enriquecidas que operan constantemente a través de diversas plataformas.

La biblioteca de JavaFX está escrita como una API de Java, las aplicaciones JavaFX puede hacer referencia a APIs de código de cualquier biblioteca Java. Por ejemplo, las aplicaciones JavaFX pueden utilizar las bibliotecas de API de Java para acceder a las capacidades del sistema nativas y conectarse a aplicaciones de middleware basadas en servidor.

La apariencia de las aplicaciones JavaFX se pueden personalizar. Las Hojas de Estilo en Cascada (CSS) separan la apariencia y estilo de la logica de la aplicación para que los desarrolladores puedan concentrarse en el código. Los diseñadores gráficos pueden personalizar fácilmente el aspecto y el estilo de la aplicación através de CSS.

Se uede desarrollar los aspectos de la presentación de la interfaz de usuario en el lenguaje de scripting FXML y usar el código de Java para la aplicación lógica.

Si se prefiere diseñar interfaces de usuario sin necesidad de escribir código, entonces, se puede utilizar JavaFX Scene Builder. Al diseñar la interfaz de usuario con javaFX Scene Builder el crea código de marcado FXML que pueden ser portado a un entorno de desarrollo integrado (IDE) de forma que los desarrolladores pueden añadir la lógica de negocio.

Así pues por un lado tendremos:

- Un fichero fxml que representará la vista
- Un posible fichero CSS para los estilos de esa vista
- Ficheros java para el modelo y para la vista-controlador

Pero ¿ qué es fxml ?

FXML es un lenguaje de marcado declarativo basado en XML para la construcción de una interfaz de usuario de aplicaciones JavaFX. Un diseñador puede codificar en FXML o utilizar JavaFX Scene Builder para diseñar de forma interactiva la interfaz gráfica de usuario (GUI). Scene Builder genera marcado FXML que pueden ser portado a un IDE para que un desarrollador pueda añadir la lógica de negocio.

Hojas de estilo en cascada CSS

Ofrece la posibilidad de aplicar un estilo personalizado a la interfaz de usuario de una aplicación JavaFX sin cambiar ningún de código fuente de la aplicación. CSS se puede aplicar a cualquier nodo en el gráfico de la escena JavaFX y se aplica a los nodos de forma asincrónica.

JavaFX también puede aplicar estilos CSS fácilmente asignados a la escena en tiempo de ejecución, haciendo que una aplicación para cambiar dinámicamente.

Controles de interfaz de usuario(UI Controls)

Veamos algunos ejemplos de controles del interfaz de usuario:



La mejor forma de entender seguramente sea mediante ejemplo y seguiremos ese procedimiento. De cualquier forma, si se prefiere tener acceso a toda la documentación, se puede ver en los enlaces:

https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm

<https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html>

y un interesante tutorial (en inglés) que ya no es de oracle:

<http://tutorials.jenkov.com/javafx/index.html>

Primer proyecto de Ejemplo de JavaFX

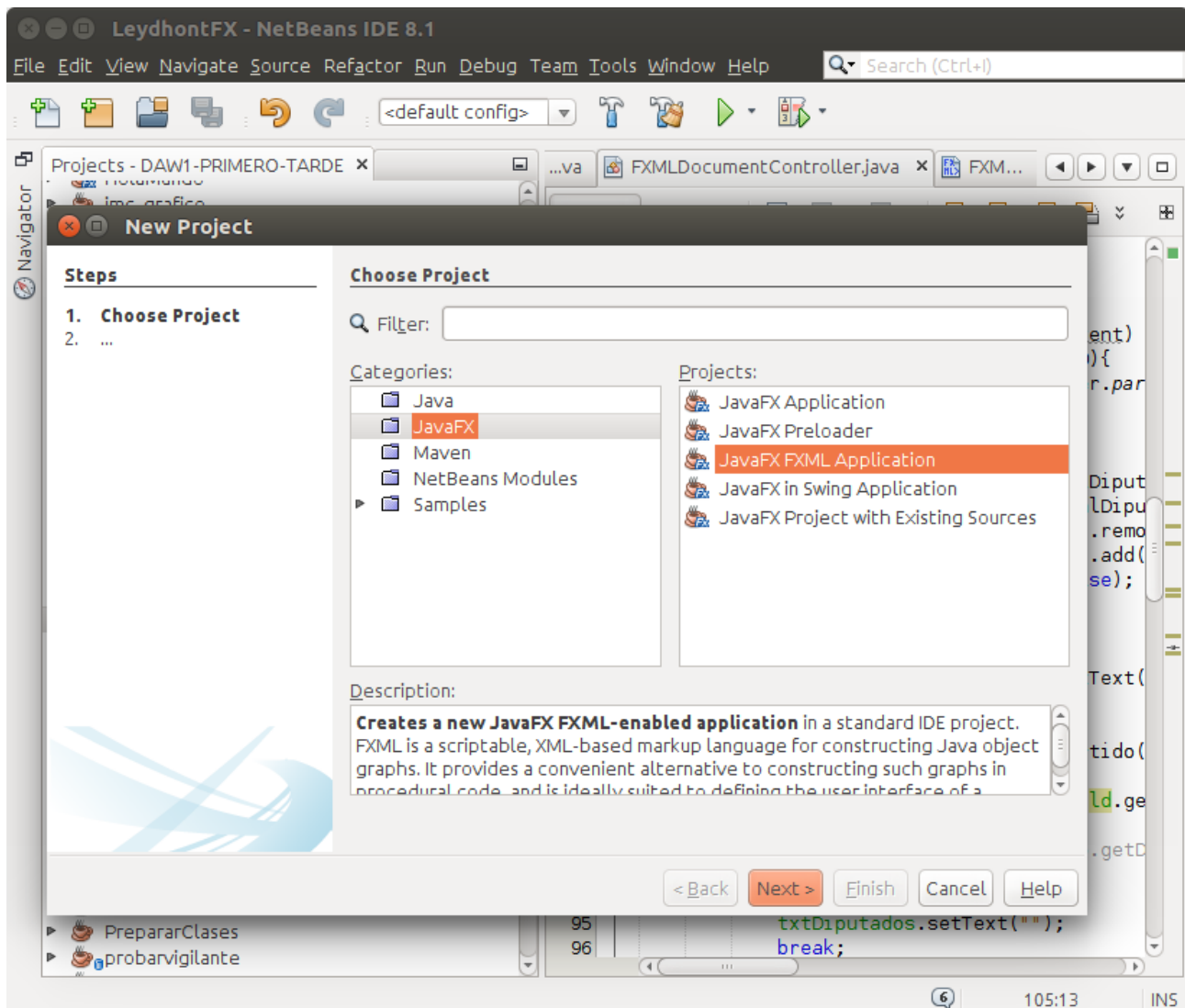
Vamos a precisar que el IDE tenga soporte para JavaFX, posiblemente una opción rápida sea instalar el paquete unificado que da Oracle de jdk y netbeans. Ya trae instalado y configurado JavaFX: <http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html>

Adicionalmente se precisará el Scene Builder. En este caso se propone el de Gluon: <http://gluonhq.com/products/scene-builder/#download>

Se propone elegir la opción de “Executable Jar” por ser independiente de la plataforma (windows-linux-mac) y no ser necesarios permisos de administrador

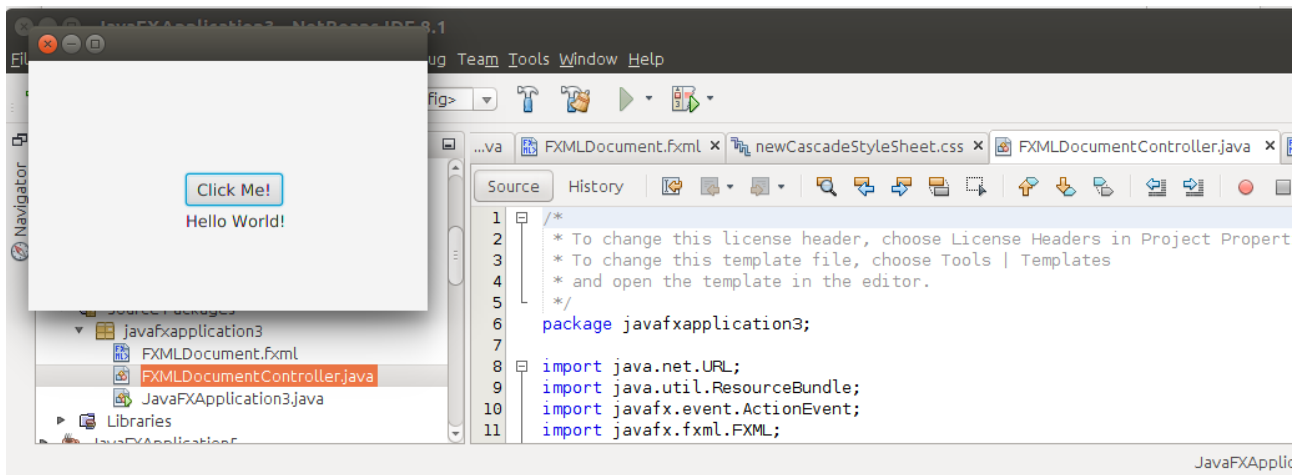
Una vez instalado todo el procedimiento es desde Netbeans:

File → New Project → JavaFX → JavaFX FXML Application



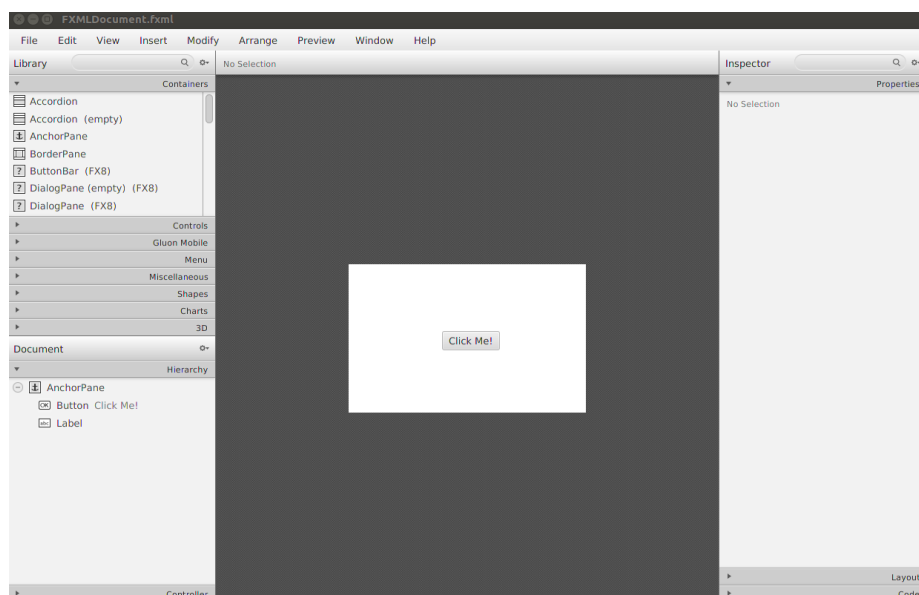
Al finalizar la creación del proyecto ya nos ha creado una miniaplicación funcional. Compuesto por tres ficheros: el fichero con el main, el fichero para la vista: (extension .fxml) y el fichero para el controlador (FXMLDocumentController.java)

Si ejecutamos el proyecto nos muestra una ventana con un botón “click me”



Paramos la ejecución.

Ahora, para la personalización de la vista pulsamos doble click sobre el fichero con extensión: .fxml y nos abrirá el Scene Builder (también: botón derecho->open)



Observamos que hay un AnchorPane, un button y un label

Desde aquí con una interfaz gráfica podemos eliminar, agregar, editar todos los contenedores y controles que queramos en nuestra vista

Si en lugar de haber elegido open hubiéramos hecho: botón derecho sobre el fichero .fxml → Edit

nos habría abierto el fichero .fxml directamente, sin editores gráficos:

```
<?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="javafxapplication3.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>
```

Como se puede observar, no difiere de otros XML que pudiéramos haber visto anteriormente.

Interesante observar en el código xml es el atributo: **onAction**

```
<Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleButtonAction" fx:id="button" />
```

De tal forma se está estableciendo en la vista que el método: `handleButtonAction()` del `FXMLDocumentController.java` se encarga de manejar el evento del botón. Eso significa que podemos desarrollar nuestro controlador por un lado y nuestra vista por otro. Y simplemente agregando el atributo a la línea xml de la vista que nos corresponda ya quedarían enlazados. Independizando mucho la vista del controlador. Observar que para referenciarlo utilizamos el símbolo: “#” en el nombre del método: **onAction="#handleButtonAction"**

También es interesante que observemos como se le dice a la vista cuál es el fichero controlador que le corresponde:

```
<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320" xmlns:fx="http://javafx.com/fxml/1" fx:controller="javafxapplication3.FXMLDocumentController">
```

Mediante el atributo: **fx:controller** se establece el nombre del fichero .java que será el controlador para esta vista

Veamos ahora el fichero controlador que nos ha generado:

```
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;

/**
 *
 * @author carlos
 */
public class FXMLDocumentController implements Initializable {


    @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 rb) {
        // TODO
    }

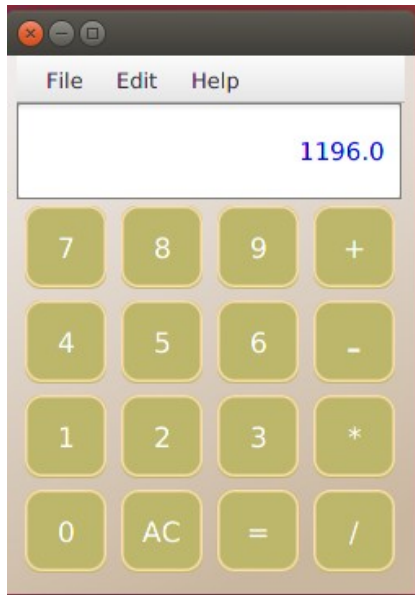
}
```

Mediante la anotación: @FXML establecemos que estamos con un objeto FXML y es en este fichero donde vemos el método que gestiona los eventos del botón: handleButtonAction()

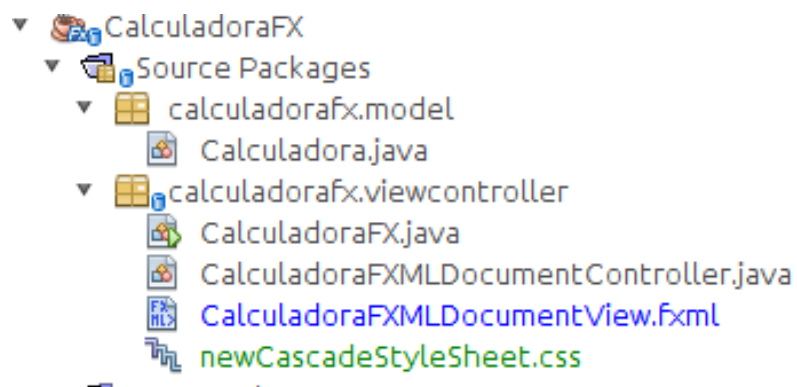
 **Práctica 3:** Realizar la aplicación anterior, modificándola para que Muestre nuestro nombre completo cuando hacemos click en el ratón. Tomar captura de pantalla de la aplicación mostrando cuando pulsamos el botón

Segundo proyecto: Calculadora

Vamos a realizar la siguiente calculadora:



En esta aplicación tendremos una idea clara de la separación del modelo: MVC



Se ha elegido crear un paquete para poner los ficheros java del modelo
y otro paquete para guardar la vista y controlador

Calculadora.java contiene la clase publica Calculadora que funciona como una calculadora antigua. Esto es, recibe un número y lo reserva, luego un operador y lo reserva, el segundo número lo reserva. Acto seguido al pulsar el símbolo “=” obtiene el resultado de la operación. De igual

manera si en lugar de pulsar el “=” pulsa otro operador obtiene el cálculo anterior y lo convierte en el primer operando para la siguiente operación.

Ej. si se introduce la siguiente secuencia de comandos/instrucciones:

2 + 3 * 7 =

en el momento en que se pulsa el símbolo: “*” ejecuta la operación: 2+3 → 5

y luego toma ese número para que sea el primero operando:

5 * 7 =

mostrando en pantalla 35

Una posible implementación de los métodos del controlador
(CalculadoraFXMLDocumentController.java) podría ser:

```
@FXML
private void operando(MouseEvent event) {
    Button btn = (Button)event.getSource();
    int operando = Integer.parseInt(btn.getText());
    calc.cargarNumero(operando);
    txtResultado.setText(calc.getResultado());
}

@FXML
private void operador(MouseEvent event) {
    Button btn = (Button)event.getSource();
    String op = btn.getText();
    calc.operar(op);
    txtResultado.setText(calc.getResultado());
}

@FXML
private void igual(MouseEvent event) {
    Button btn = (Button)event.getSource();
    String op = btn.getText();
    calc.operar(op);
    txtResultado.setText(calc.getResultado());
}

@FXML
private void limpiar(MouseEvent event) {
    calc.limpiar();
    txtResultado.setText(calc.getResultado());
}
```

Donde calc es un objeto de tipo Calculadora creado en la inicialización:

```
public void initialize(URL url, ResourceBundle rb) {
    calc = new Calculadora();
}
```

Es fácil observar que este objeto Calculadora funciona perfectamente tanto para una aplicación de consola como para una aplicación gráfica. Por ejemplo, en consola la ejecución de las instrucciones que nombramos antes: 2 + 3 * 7 =

```
calc.limpiar();  
calc.cargarNumero(2);  
calc.operar("+");  
calc.cargarNumero(3);  
calc.operar("*");  
calc.cargarNumero(7);  
calc.operar("=");  
calc.getResultado();
```

el método limpiar() elimina todo lo que pudiera haber en memoria cargado.

cargarNumero() introduce un operando

operar() realiza la operaciones pendientes y establece el nuevo operador.

Así pues Calculadora.java es el **modelo** de nuestra aplicación perfectamente separado y portable para cualesquier interfaz gráfica que quisiéramos utilizar

Como se puede observar en: CalculadoraFXMLDocumentController.java lo que se hace en el controlador es enlazar el elemento gráfico vista FXML con el modelo. Por ejemplo:

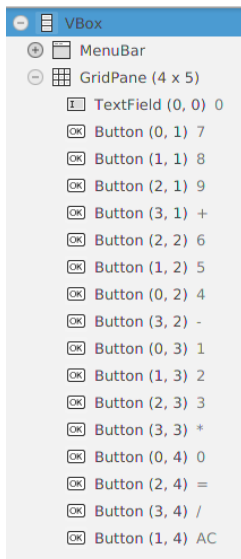
```
@FXML  
private void operando(MouseEvent event) {  
    Button btn = (Button)event.getSource();  
    int operando = Integer.parseInt(btn.getText());  
    calc.cargarNumero(operando);  
    txtResultado.setText(calc.getResultado());  
}
```

nos muestra que toma de la vista el número que aparece en el botón mediante parseInt lo convierte en entero y luego se lo pasa al modelo mediante: calc.cargarNumero()

finalmente muestra lo que devuelve el modelo en pantalla en el txtResultado que ha creado la vista a tal efecto.

En la vista del navegador del proyecto que se ha mostrado antes también se observa un fichero .css que nos sirve para dar los estilos a nuestra aplicación

Pasos para crear la vista



Dentro de Scene Builder introducimos un contenedor VBox que a su vez va a tener un control MenuBar y un contenedor GridPane

Vbox es un layout que posiciona todos sus hijos (los componentes) en una fila vertical.

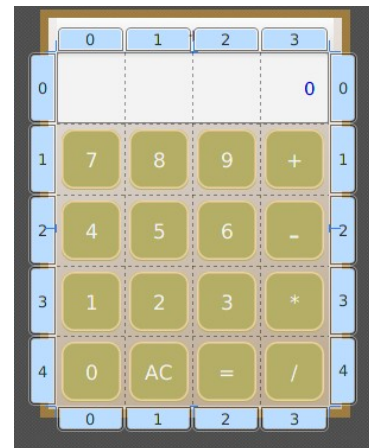
Luego arrastramos un control MenuBar hacia el VBox

de esta forma será el primer elemento que muestre la aplicación, ya que típicamente se suelen mostrar los menu en la parte alta de la interfaz de usuario (ui)

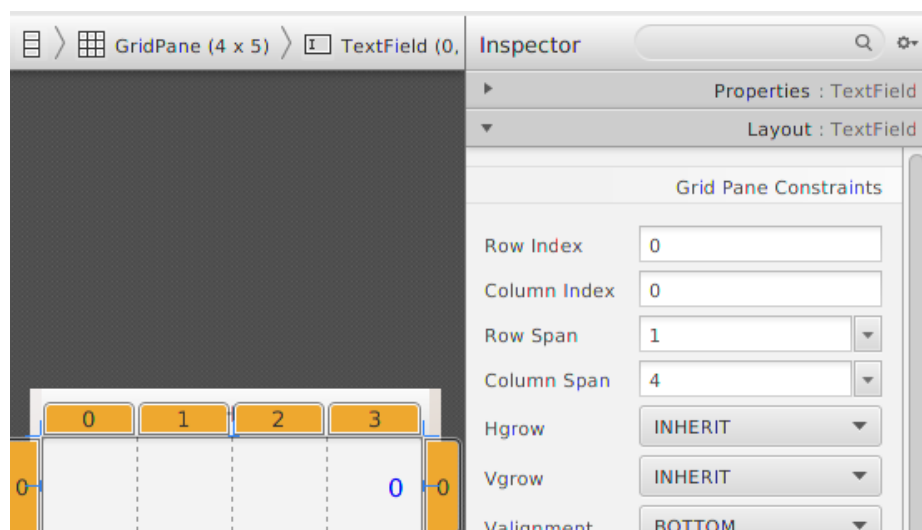
MenuBar se redimensiona automáticamente al ancho del VBox con los elementos menuitem que se vayan introduciendo /quitando

Posteriormente arrastramos un GridPane dentro del VBox

Este GridPane lo hacemos de 5filas y 4columnas

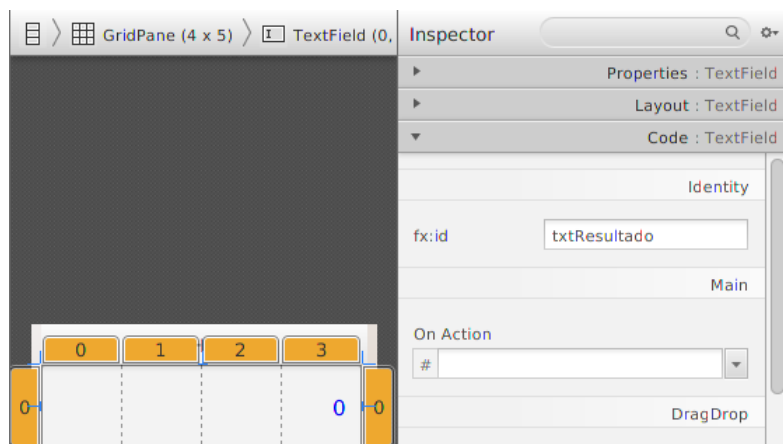


Como ejemplo de como establecer los elementos dentro del grid veamos la siguiente imagen:



En la imagen observamos que el elemento que hemos arrastrado un TextField a la posición 0,0 del grid. Y le hacemos un Column Span de 4 de esa forma la caja de texto para el resultado tomará toda la primera fila del grid

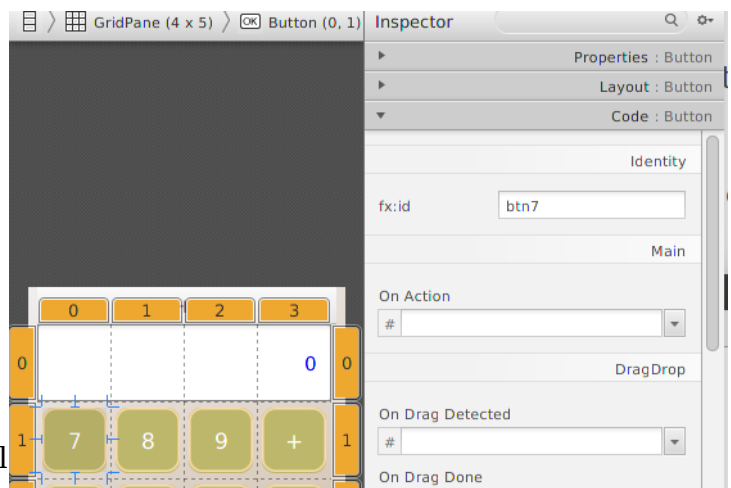
En la parte de code le ponemos el id que luego podremos utilizar tanto en el controlador como en el fichero css. En esta ocasión se ha elegido como id: txtResultado



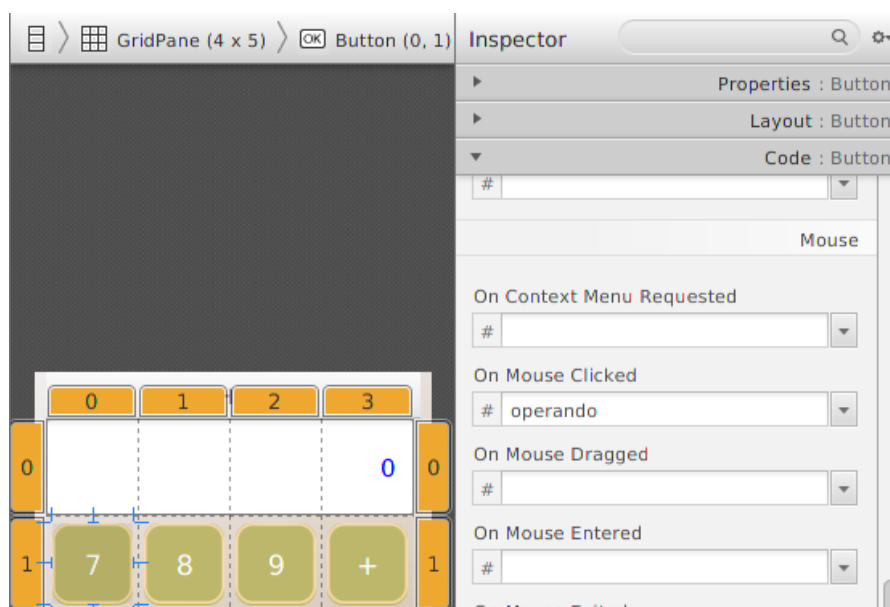
Los botones con números se ha elegido como id: btnNum

así el botón para el número 7 nos queda:

id: btn7



Adicionalmente establecemos para el botón el método que va a lanzar cuando sea pulsado:



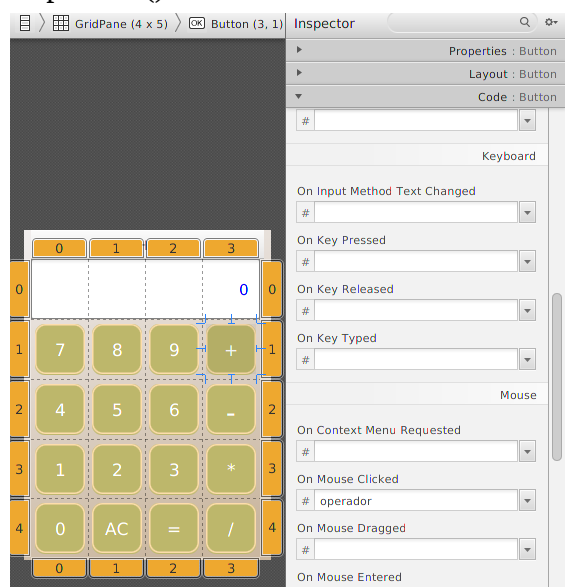
observar que no escribimos los paréntesis en el nombre del método y que lo hemos establecido para el evento: On Mouse Clicked

Para los operadores: “+”, “-”, “*”, “/”, “=” se ha elegido como identificador respectivamente:

btnMas, btnMenos, btnPor, btnDividir, btnIgual

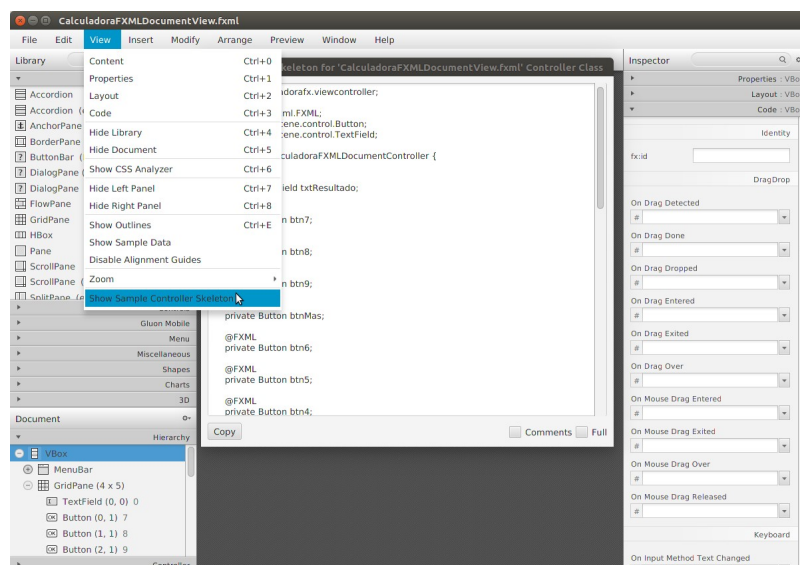
y enlazarles el mismo método a todos ellos: operador()

Como ejemplo vemos para el “+”:

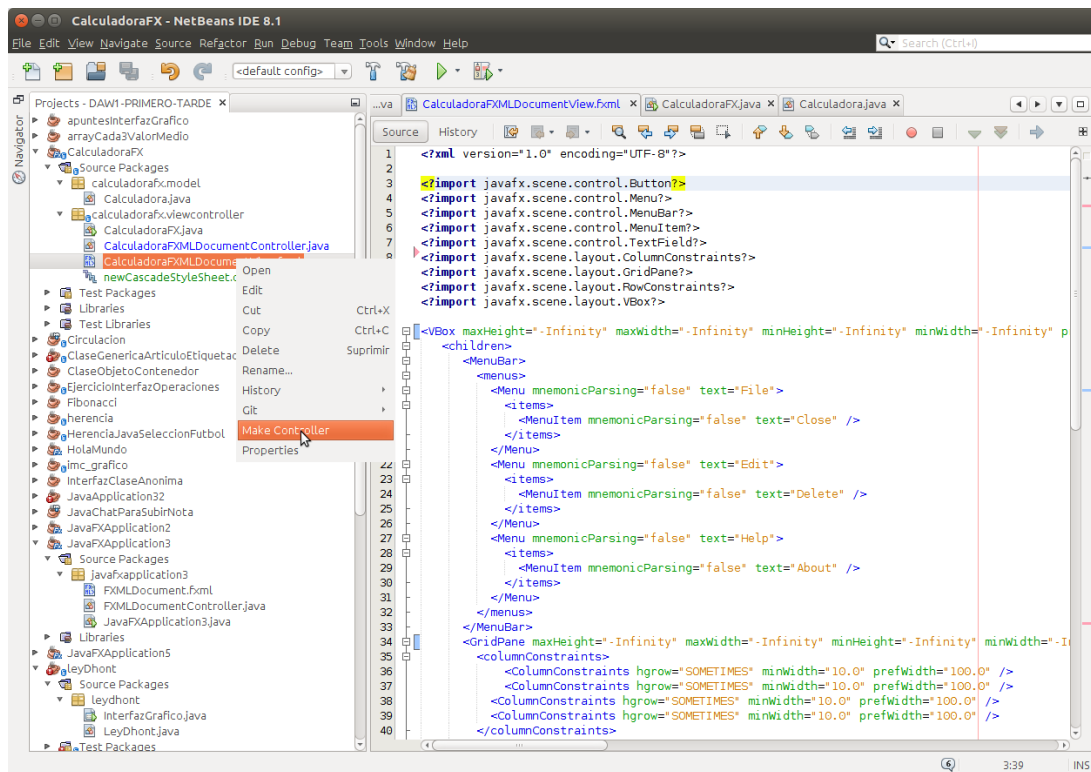


Se puede ver como nos quedaría la estructura de un fichero .java controlador

view → Show
sample controller
skelletion



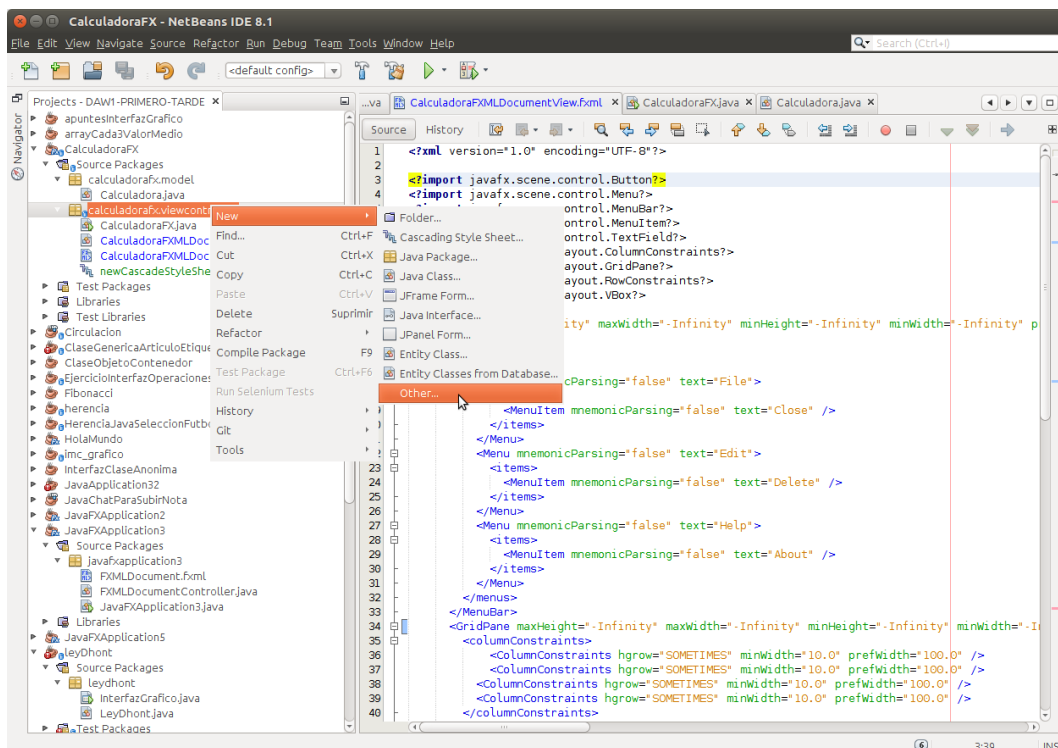
Seleccionar ese texto copiarlo y llevarlo al fichero controller puede sernos muy útil. De cualquier forma tenemos otra alternativa, posiblemente mejor. Nos ponemos en netbeans sobre el fichero .fxml que nos ha sido generado en el Scene Builder → botón derecho → Make controller

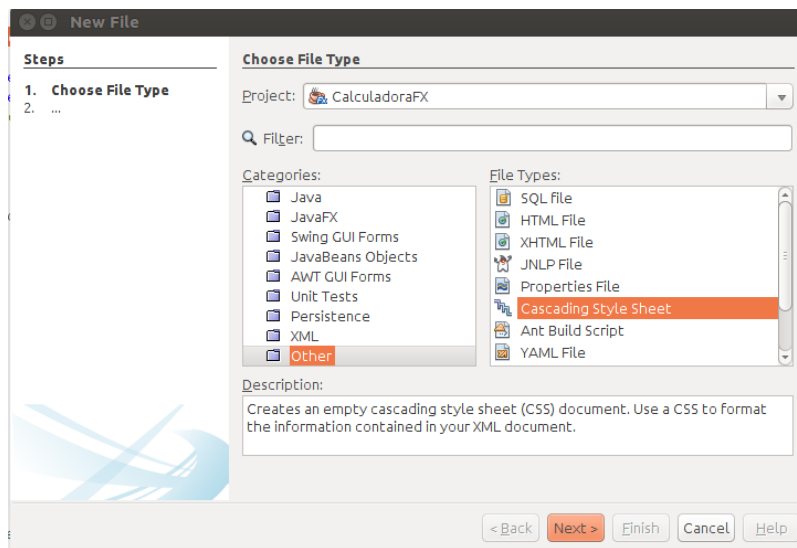


Nos falta ponerle estilos a la aplicación.

Para ello desde Netbeans botón derecho sobre el paquete donde queremos que quede el fichero:

botón derecho → New → Other → Other → Cascading Style Sheet

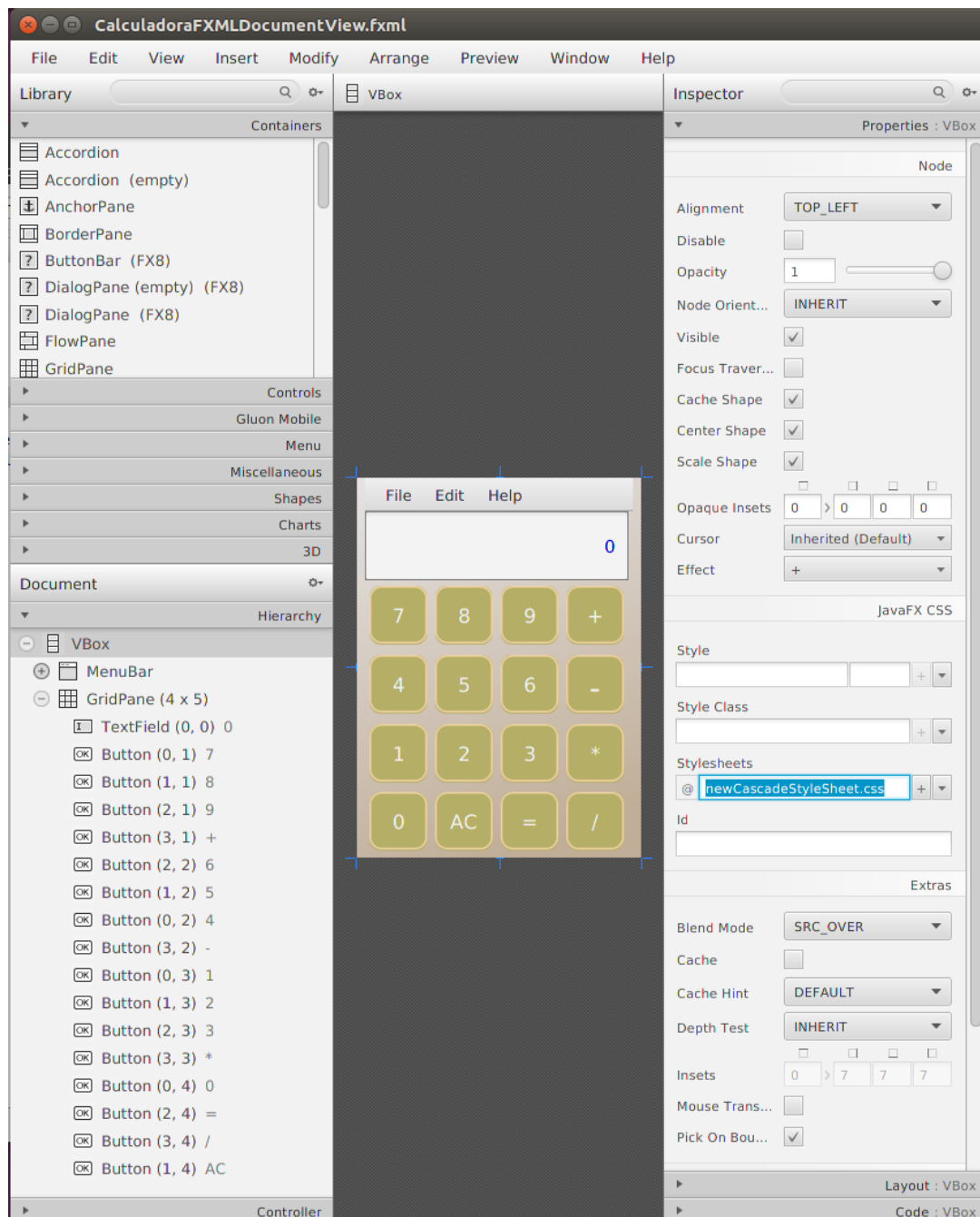




Una vez creado lo enlazamos con la vista desde el Scene Builder:

Nos ubicamos en el elemento raíz (en el caso del ejemplo sería el VBox), en properties buscamos donde dice Stylesheets y allí usamos el botón “+” que nos abrirá una ventana para seleccionar el fichero.

Nota: Para ver los efectos que nos queda en el CSS podemos pulsar **CTRL+P** (**Preview** → **Show Preview in Window**)



Ahora bien, este fichero CSS utiliza instrucciones específicas, ahora detallaremos alguna, pero lo mejor es remitirse a la documentación CSS que se especificó antes

Podemos usar selectores genéricos:

`.button{}` → nos aplicará estilos a todos los botones

`.button:hover{}` → comportamiento cuando el puntero esté sobre un botón

`.button:pressed{}` → comportamiento cuando pulsamos sobre el botón

`.root{}` → raíz del documento. En nuestro caso coincidiría con el VBox

`.text-field{}` → estilos a todos los textfield (en nuestro caso a la caja de resultados)

También podemos acceder mediante el id que hemos generado para luego usar desde código. Así por ejemplo, podemos acceder específicamente al botón con el número 7 mediante:

`#btn7{}`

Respecto a las instrucciones que podemos utilizar son parecidas a cualesquier CSS. La diferencia es que suelen tener: “-fx” como prefijo. Así por ejemplo, para poner un fondo que tenga un gradiente y un padding como en la imagen de ejemplo. Podemos escribir:

```
.root{  
  -fx-background-color: linear-gradient(from 0% 0% to 100% 200%, repeat, #F1EEEE 0%, #CAB7A0 50%);  
  -fx-padding: 0 7px 7px 7px;  
}
```

Otras instrucciones que se han usado en alguna parte del ejemplo son:

`-fx-border-color` → color del borde

`-fx-background-radius` → curvatura en los límites del fondo del elemento

`-fx-border-radius` → curvatura en el borde del elemento

`-fx-text-fill` → color de la letra

`-fx-font` → tamaño de la letra, negrita etc. Ej: `-fx-font: 10px bold;` → 10px y negrita

`-fx-border-width` → tamaño del border

`-fx-scale-x` → escalar el objeto en la coordenada x

`-fx-scale-y` → escalar el objeto en la coordenada y

`-fx-font-size` → tamaño de la letra


Para terminar vamos a fijarnos en un trozo de código que pusimos antes del controlador:

```
@FXML
private void operando(MouseEvent event) {
    Button btn = (Button)event.getSource();
    int operando = Integer.parseInt(btn.getText());
    calc.cargarNumero(operando);
    txtResultado.setText(calc.getResultado());
}
```

Fijarse que podemos saber que objeto ha desencadenado el evento mediante:

`event.getSource()`

de esa forma en el código mostrado se obtiene el número del botón y se lo podemos pasar al modelo
(`calc.cargarNumero(operando)`)

 **Práctica 4:** Realizar la aplicación de calculadora descrita. Tener en cuenta que cuando se pasa el ratón encima de un botón este cambia de color y se remarca el borde. Cuando se presiona el botón toma el color de fondo y escala ligeramente el botón en x, y