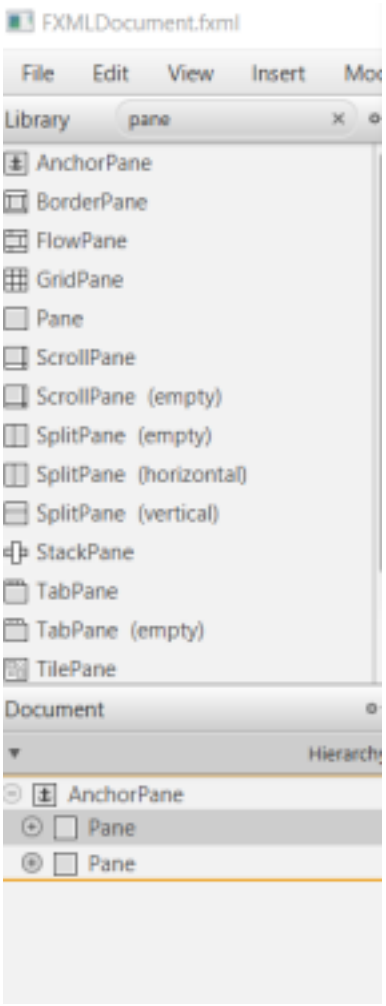
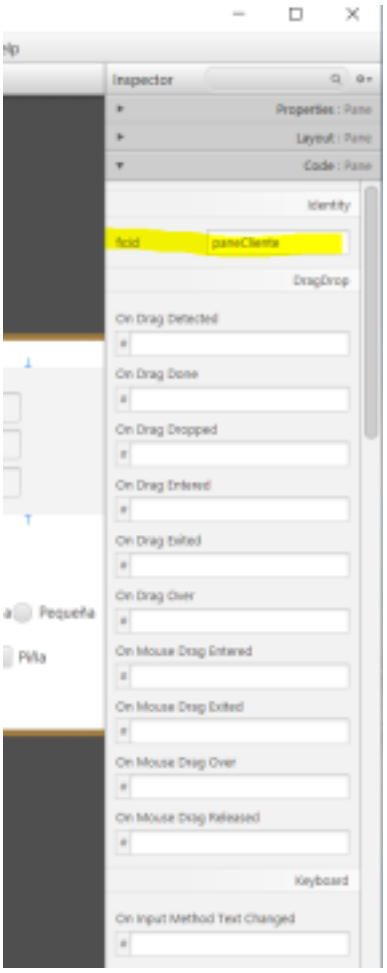
1. Pane o Contenedor

Crearemos PANE, para distribuir los elementos de nuestra interfaz (botones, cajas de texto…) y así nos permitirá una mejor organización de los mismos

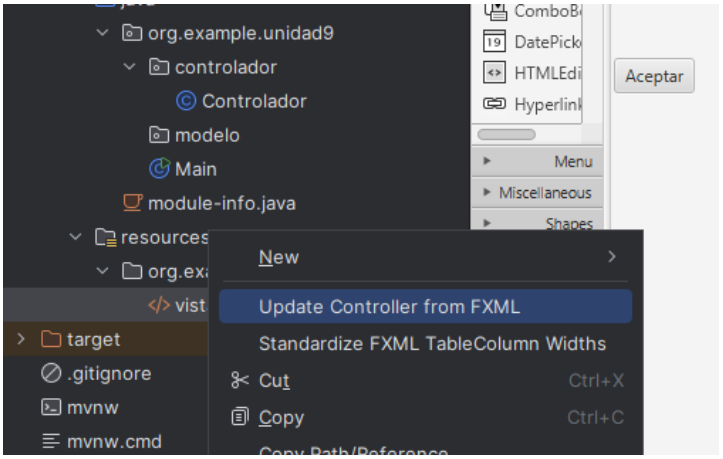


En javaFx siempre debemos ponerle un identificador (parte derecha , apartado Code) a los componentes para que pueda pasarlo al Controller.java



IMPORTANTE: INSTALAR PLUGIN FMXLManger. (FILE→Settings→Plugins (Buscar en Marketplace el plugin FXMLManager)

Cuando guardamos el documento FXML en Scene Builder, y nos vamos a Intellij, debemos seleccionar el archivo FXML (que normalmente estará en el paquete resources) → pulsaremos botón derecho sobre el → seleccionando “Update controller from FXML” para actualizar todos los elementos incluidos en Scene Builder (que le hayamos puesto id).



Todos los elementos que hayan sido creados y especificado id, nos aparecerán en

Controller.java con etiqueta @FXML, ejemplo a continuación

Volvemos a Scene Builder para continuar creando nuestra interfaz gráfica

1.1.Deshabilitar un PANE

En el método initialize, que es el primero que ejecuta, pondremos los valores que debe tener nuestro FXML al arrancar, por ejemplo deshabilitar un PANE completo

En el ejemplo de nuestra tarea el panePizza lo deshabilitaremos al arrancar y cuando introduzca todos los datos del cliente y le de al botón Guardar, lo habilitaremos.

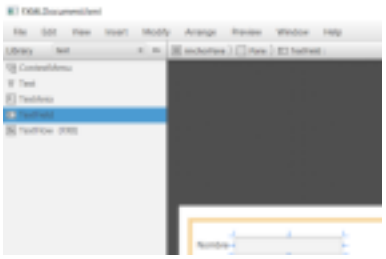
```
@Override
public void initialize(URL url, ResourceBundle rb) {
panePizza.setDisable(true);
}
```

Y en el método asociado al botón que se creará, tendremos que habilitarlo si ha cumplimentado todos los datos de cliente:

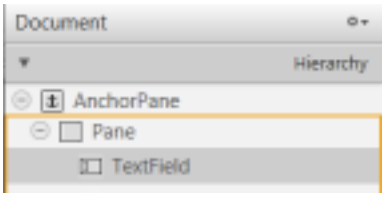
```
panePizza.setDisable(false);
```

2. Textfield – Cajas de Texto

Como sabemos nos servirán para introducir contenido, buscamos textField, seleccionamos, para arrastras y soltar en la interfaz



Introducimos dentro de nuestro Pane, y en la parte izquierda inferior nos irá apareciendo la estructura que vamos completando



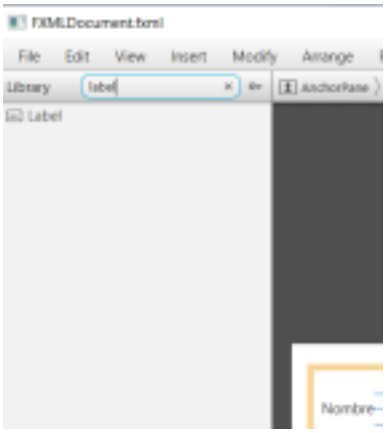
A cualquier elemento, en este caso textfield, le pondremos idetificador para que pueda pasarlo al Controller.java (parte derecha , apartado Code. Igual que el apartado anterior.

3. Label o etiquetas

Que uilizaremos para describir un campo de texto, o cualquier otro elemento que lo necesite

A las etiquetas no le pondremos identificador, puesto que en principio no necesitaremos programar algo sobre ese elemento en el Controller.java

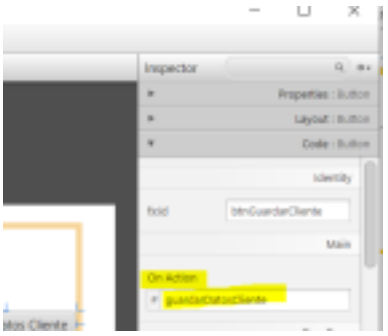
Buscaremos dicho elemento para arrastras y soltar



4. Button – botones

Podriamos decir que puede tener un tratamiento especial, puesto que al pulsarlo querremos que haga algo, que tenga una acción que lanzar.

Por ello, además del id, indicaremos el nombre del método que se lanzará en el apartado “On Action”



Al guardar en Scene Builder y volver a crear desde Intellij y generar con el " Update controller from FXML", tendremos el método para implementarlo, con la etiqueta @FXML generado de forma automática desde Scene Builder

```
@FXML
private void guardarDatosCliente(ActionEvent event) {
```

4.1.Métodos (asociados a Evento o Acción)

En este caso, al pulsar el botón guardar, debemos implementar el código que el Controller nos ha incluido con etiqueta @FXML y con nombre de método el indicado en Apartado "Code" → On Action

4.1.1. Alert

En este caso, crearemos un objeto Alert, que consiste en una ventana que se lanzará en determinadas situaciones que consideremos que se necesite avisar al usuario de algo que tenga que subsanar.

Indicando que tipo de Alert será, en este caso informativo

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

AlertType:

Titulo del objeto Alert:

alert.setTitle("ERROR");

Mensaje de cabecera del objeto Alert:

alert.setHeaderText("Mensaje de error");

Mensaje/contenido:

alert.setContentText("Debe introducir el NOMBRE de cliente");

Mostrar mensaje y espera a confirmar – pulsar Aceptar:

alert.showAndWait();

4.1.2. Acceder al contenido de los TextField

nombreTextField.getText()

Por ejemplo:

tfNombre.getText()

Si queremos comprobar si este está vacío para mostrarle un objeto Alert en caso de no haberlo cumplimentado, utilizaremos el método isEmpty, que devuelve true si es vació y false si tiene algún valor introducido

tfNombre.getText().isEmpty()

5. ComboBox o desplegable

Seleccionamos este elemento para arrastrarlo sobre el Pane indicado
Lo característico de este elemento es que necesitamos cargar en el controller.java los valores que podría tener
Para ello, en el método initialize (que es el método inicial que se lanzará, nada más cargar la pantalla) es donde incluiremos los valores iniciales que debe tener nuestro comboBox, o cualquier información de partida de nuestra pantalla
Especificaremos con nombre de nuestro objeto cbTipo (especificado en id), invocaremos el método getItemos (para obtener los elementos de nuestro combo) y método addAll, que permitirá añadir lo que debe incluir nuestro combo

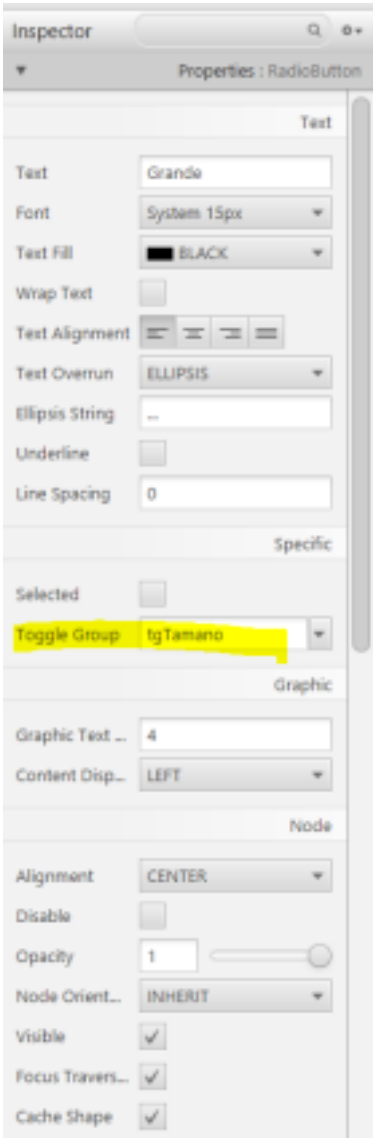
cbTipo.getItems().addAll("Margarita", "4 quesos", "BBQ", "Carbonara", "Hawaina");

Y si queremos incluir un prompt, mientras selecciona una opción, aquí tenemos un ejemplo

cbTipo.setPromptText("Introduce tipo pizza");

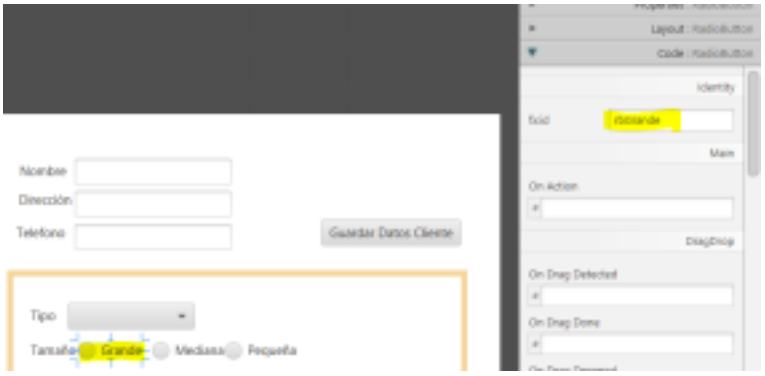
6. RadioButton o botones de radio

En estos elementos solo puede seleccionarse uno a la misma vez, por lo que suele crearse un Toogle Group, al que pertenece un grupo de RadioButton y este se especifica en Properties del elemento en característica Tootle Group, donde elegiremos un nombre y lo incluiremos en dicha característica



Y como el resto de componentes de JavaFX le daremos un identificador a cada RadioButton. De manera que al inicializar también especifiquemos por defecto un valor inicial marcado.

No necesitamos etiqueta porque en la propiedad Text (apartado Properties) pondremos el nombre que aparecerá a la derecha del RadioButton



Estableciendo en el Controler.java que será este el valor por defecto al arrancar el programa:

```
@Override
 public void initialize(URL url, ResourceBundle rb) {
 rbGrande.setSelected(true);
}
```

7. Checkbox o casilla verificación

Este componente tiene como característica que puede seleccionarse los que se deseen, a diferencia de RadioButton, estos últimos no son excluyentes Como el resto de componentes de JavaFX le daremos un identificador a cada CheckBox

No necesitamos etiqueta porque en la propiedad Text (apartado Properties) pondremos el nombre que aparecerá a la derecha del RadioButton

No necesitaremos marcar en el Controler.java cual será este el valor por defecto al arrancar el programa, puesto que estarán deseleccionados

Habrá una única etiqueta que identifique al grupo tanto de RadioButton como de CheckBox (una para cada uno) que si bien no necesitará identificará tendrá un Text que colocaremos delante del grupo o donde consideremos que sea representativo del grupo