

# 1. Refuerzo de Java

---

**Parte V. Más conceptos de JavaFX**

**Programación de Servicios y Procesos**

Nacho Iborra  
Álvaro Pérez  
Javier Carrasco



Esta obra está bajo una [Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-nc-sa/4.0/).

## Tabla de contenido

<b>13. Algunos conceptos más útiles.....</b>	<b>3</b>
13.1. Añadir imágenes a una aplicación.....	3
13.2. Evento cerrar ventana.....	3
13.3. Diálogos integrados.....	4
13.4. Cambiar la apariencia predeterminada con CSS.....	6
13.5. Aplicaciones con múltiples vistas.....	8
13.6. Usando gráficos.....	9
<b>14. Ejemplo: HealthyMenu.....</b>	<b>12</b>
14.1. Diseño de la vista principal.....	12
14.1.1. Definiendo la tabla.....	12
14.1.2. Definiendo la parte inferior.....	14
14.2. Cargar datos en la tabla.....	17
14.2.1. Leer/Guardar datos de/a un fichero.....	20
14.3. La vista del gráfico.....	21
14.4. Modularidad.....	22
14.5. Añadir eventos.....	23
14.5.1. Añadir un nuevo alimento a la tabla.....	23
14.5.2. Cambio a la vista gráfico.....	24
14.5.3. Cambio hacia la vista principal.....	25

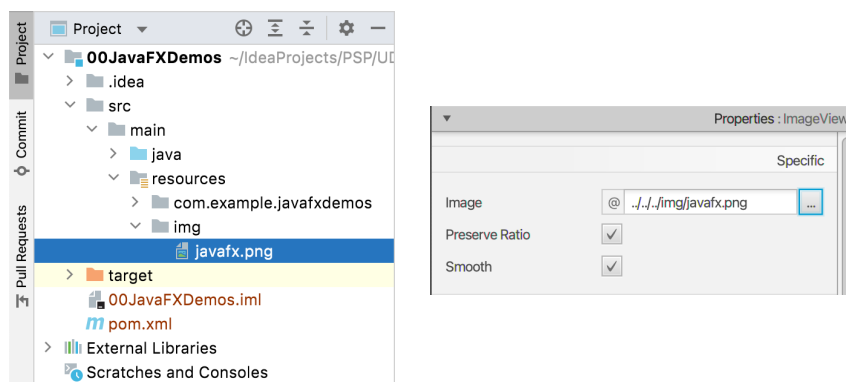
## 13. Algunos conceptos más útiles

Se han cubierto algunos de los conceptos básicos de la creación de una aplicación JavaFX, como crear el proyecto, *layouts* y controles y definir eventos. Pero hay más opciones más que se suelen utilizar y pueden ser útiles para crear una aplicación de escritorio profesional. Se hablarán de ellas a continuación.

### 13.1. Añadir imágenes a una aplicación

Es habitual añadir algunas imágenes a la aplicación para hacer más atractivos algunos controles (o el aspecto general). Para hacer esto, se utilizarán las clases ***Image*** e ***ImageView*** (de l paquete *javafx.scene.image*). El primero permite cargar una imagen desde un archivo (normalmente en la carpeta raíz de la aplicación, aunque puede estar en cualquier otro lugar). La clase *ImageView* adapta la imagen cargada a algo que la pueda dibujar en la aplicación (un *drawable*).

Para añadir una imagen a una aplicación FXML, simplemente arrastra el control *ImageView* a la escena. Después, escribe la ruta relativa (al archivo FXML) de la imagen. Utiliza la carpeta *resources* para añadir las imágenes.



Se puede cambiar la imagen que se muestra en el control *ImageView* en código, configurando un nuevo objeto *Image* con otra ruta indicar la URL de la imagen:

```
imgFX.setImage(new Image("file:src/main/resources/img/javafx.png"));
```

### 13.2. Evento cerrar ventana

Si el usuario intenta cerrar la ventana de la aplicación con el botón de cierre de la barra superior (o utilizando *Alt+F4* o cualquier otro atajo), se omiten los esfuerzos realizados para intentar guardar los datos antes de cerrar, u obtener una confirmación del usuario antes de salir del programa.

Pero no te preocupes. Hay una forma de capturar este evento y redirigirlo a un método de control. Sin embargo, será necesario iniciar la aplicación de una manera diferente para poder obtener una referencia al controlador de la clase principal de la aplicación:

```

@Override
public void start(Stage stage) throws IOException {
    FXMLLoader loader = new FXMLLoader(getClass().getResource("hello-view.fxml"));
    Parent root = loader.load();

    HelloController controller = (HelloController)loader.getController();
    controller.setOnCloseListener(stage);

    stage.setTitle("Hello World");
    stage.setScene(new Scene(root, 600, 400));
    stage.show();
}

```

Observa que una vez que se obtiene una referencia al controlador, se puede llamar al método creado ***setOnCloseListener()*** y pasarle el ***stage*** como argumento. Así es como se implementa este método en el controlador:

```

public class HelloController {
    ...
    public void setOnCloseListener(Stage stage) {
        stage.setOnCloseRequest(e -> {
            e.consume(); // This way we can prevent window from closing!
        });
    }
    ...
}

```

### 13.3. Diálogos integrados

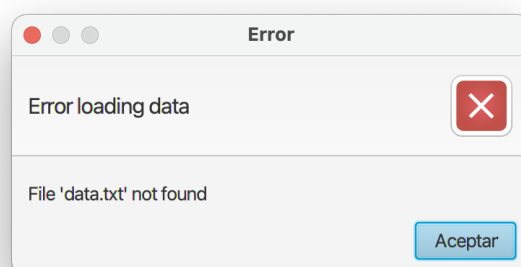
Desde JavaFX 8 (o más bien, desde JavaFX versión 8u40) existen diálogos integrados disponibles para JavaFX. Algunos de ellos muestran mensajes de información o cuadros de diálogo de confirmación. La mayoría de estos cuadros de diálogo se pueden crear desde la clase ***Alert***. Tiene métodos para definir el título, el encabezado y el contenido del diálogo, aunque todos estos mensajes son opcionales. El uso básico de esta clase es mostrar mensajes básicos, como mensajes de error o mensajes de información.

```

Alert dialog = new Alert(AlertType.ERROR);

dialog.setTitle("Error");
dialog.setHeaderText("Error loading data");
dialog.setContentText("File 'data.txt' not found");
dialog.showAndWait();

```

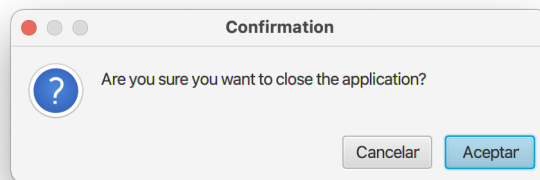


Se puede cambiar el parámetro del constructor (*AlertType.ERROR*) por cualquier otra constante de la clase *AlertType*, como *CONFIRMATION*, *WARNING*, *INFORMATION*, ... si se utiliza el cuadro de diálogo *CONFIRMATION*, se mostrarán dos botones:

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

dialog.setTitle("Confirmation");
dialog.setHeaderText("");
dialog.setContentText("Are you sure you want to close the application?");
Optional<ButtonType> result = dialog.showAndWait();

if(result.get() == ButtonType.OK){
    // Code for "OK"
} else {
    // Code for "Cancel"
}
```



Si pulsas el botón “Aceptar”, se devolverá un valor. De lo contrario (si haces clic en el botón “Cancelar”, o cierras el cuadro de diálogo), no se devolverá ningún valor. Puede almacenarse el valor en un objeto *Optional* para comprobar el resultado del diálogo.

Otro cuadro de diálogo integrado desde JavaFX 2.0 es el diálogo ***FileChooser***. Se utiliza para mostrar un cuadro de diálogo en el que el usuario pueda elegir un archivo para abrir/guardar datos desde/en en este. Para usar este diálogo, simplemente añade estas líneas al código:

```
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Open Resource File");
File selectedFile = fileChooser.showOpenDialog(stage);
```

Existe el método *showSaveDialog()* que permite que el usuario guarde datos en un archivo, así como otros métodos útiles (consulta la API para obtener más detalles).

También hay algunos otros cuadros de diálogo integrados que pueden resultar útiles. Por ejemplo, se puede personalizar el cuadro de diálogo de confirmación con más botones, y se pueden crear algunos otros tipos de cuadros de diálogo, como cuadros de diálogo de entrada de texto (con la clase *TextInputDialog*, para pedirle al usuario que introduzca algún texto), o diálogos de elección (con clase *ChoiceDialog*, para que el usuario elija entre una lista de opciones).

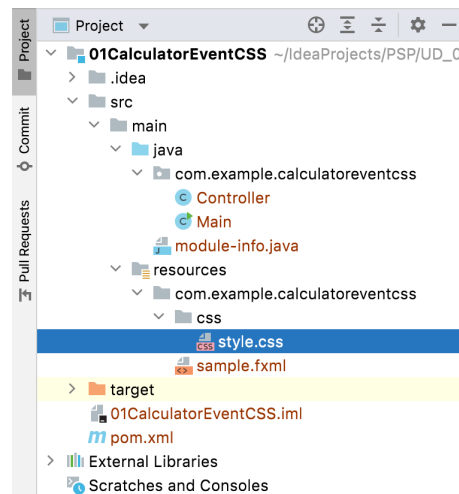
Ejemplos: <http://code.makery.ch/blog/javafx-dialogs-official/>

## 13.4. Cambiar la apariencia predeterminada con CSS

Si te fijas en alguna de las aplicaciones que se han creado hasta ahora, todas tienen un estilo anticuado, similar a las aplicaciones que se utilizaban en sistemas operativos anteriores, como Windows 95.

Pero hay una manera de cambiar esta apariencia predeterminada a una moderna, utilizando CSS, un tipo de archivo muy utilizado en el diseño web. CSS no es objeto de estudio en este módulo, ya que se asume que ya tienes una idea de qué es y cómo funciona. Se verá cómo agregar estos archivos a las aplicaciones JavaFX.

En primer lugar, deberás agregar un archivo CSS al proyecto. Como se trata de un recurso, crea una nueva carpeta dentro del paquete en *resources* llamada CSS, dentro añade un nuevo fichero, en este caso se llamará *style.css*.



A continuación, se añadirán los estilos al archivo CSS. Hay un nombre especial que se debe seguir para establecer los estilos en los controles apropiados. Por ejemplo, si se quieren aplicar estilos generales a todos los elementos de la aplicación (como color de fondo, estilos de fuente, etc) se utilizará la clase selector *root*:

```
.root
{
    -fx-background-color: lightgray;
    -fx-font-family: "Arial";
}
```

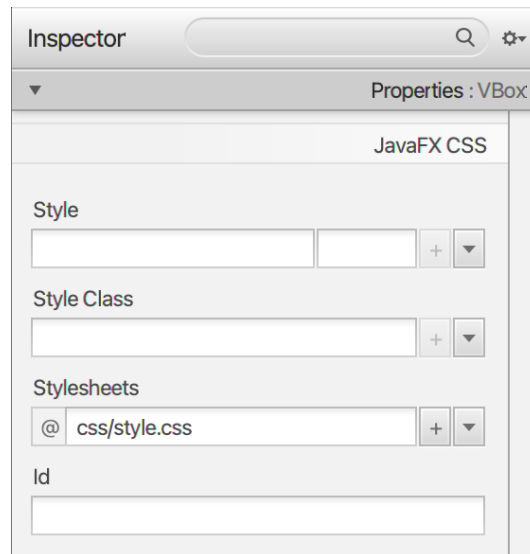
Debes tener en cuenta que cada estilo CSS comienza con el prefijo **-fx-**, seguido del nombre de estilo CSS típico (*background-color*, *font-family*, etc). También hay algunos otros selectores de clases que pueden usarse para aplicar estilos a algunos controles específicos (botones, etiquetas ...). Por ejemplo, este estilo se aplica a cada botón de la aplicación:

```
.button {
    -fx-background-color: #A22E15;
    -fx-color: white;
}
```

También se puede definir una clase o selectores de identificación propios, como:

```
.myButton { ... }
#myID { ... }
```

Después, si quieres aplicar estos estilos a algunos controles seleccionados, debes asignarles la clase o id correspondiente, desde *SceneBuilder*. También se pueden agregar propiedades CSS individuales, o incluso adjuntar un archivo de hoja de estilo CSS a un elemento y sus descendientes.



Una vez creado el fichero CSS, se puede agregar a la escena añadiéndoselo al panel principal (raíz) de la aplicación (la hoja de estilo también se aplicará a los elementos dentro de ella).

También puede configurarse en *scene* con las siguientes líneas de código en el método de inicio de la clase de la aplicación principal (la ruta al archivo CSS es relativa a la carpeta de la clase Java de la aplicación principal):

```
Scene scene = new Scene(root);
scene.getStylesheets().add("style.css");
stage.setScene(scene);
```

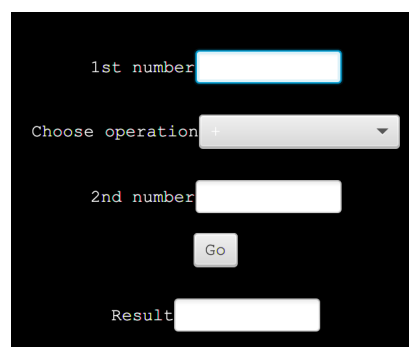
Para utilizar este sistema, el fichero *style.css* debe encontrarse dentro de la carpeta *resources*.

Referencia CSS de JavaFX:

<http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html>

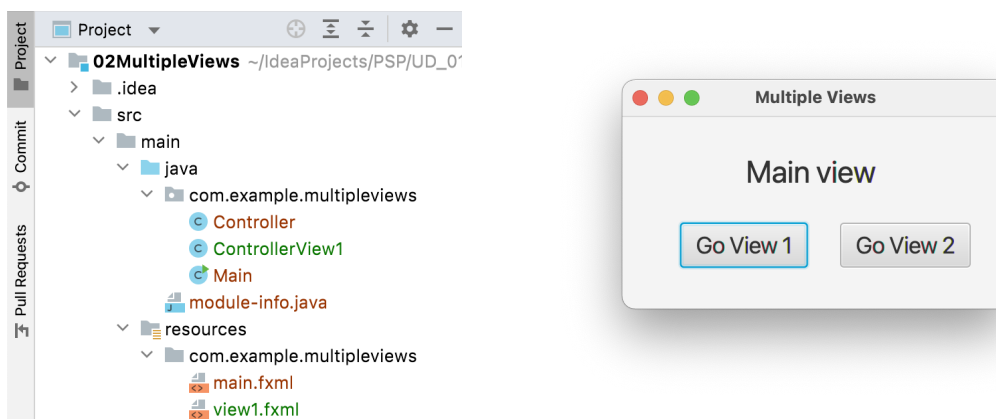
### Ejercicio 12

Crea un proyecto llamado **CalculatorEventCSS**, que será una copia del proyecto *CalculatorEvent* de la sesión anterior. Cambia su estilo predeterminado utilizando un archivo CSS que establezca el color de fondo de toda la aplicación en negro, el color del texto (para las etiquetas) en blanco y la fuente en *Courier New*. Al final, la aplicación debería verse así:

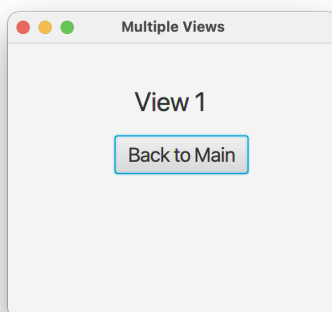


## 13.5. Aplicaciones con múltiples vistas

Suele ser útil crear aplicaciones con múltiples vistas que pueden intercambiarse fácilmente. Por ejemplo, una aplicación con una pantalla de inicio de sesión y una pantalla principal. Observa cómo funcionaría con un ejemplo: crea un proyecto JavaFX FXML nuevo llamado *MultipleViews*, y crea una vista como la siguiente. Habrán otras dos vistas que se cargarán desde esta.



Deberá crearse un nuevo archivo FXML para cada nueva pantalla. Cada archivo FXML estará asociado a una clase de controlador, que deberás crear. Las otras dos vistas se verán así:



En el método del evento *Acción* del botón “Go View 1”, solo será necesario cargar el FXML de la vista que se quiere cargar, reemplazando lo que se muestra en la ventana principal.

```
@FXML
void goToView1(ActionEvent event) throws IOException {
    Parent root = FXMLLoader.load(
        Objects.requireNonNull(getClass().getResource("view1.fxml")));

    Scene view1Scene = new Scene(root);
    Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
    stage.setScene(view1Scene);
    stage.show();
}
```



Como el código es prácticamente el mismo para todas las vistas, se podría crear una clase para administrar estos cambios de vista (tal vez con un método estático o usando herencia).

```
public class ScreenLoader {
    public static void loadScreen(String viewPath, Stage stage) throws IOException {
        // Load the view from the FXML file
        Parent view1 = FXMLLoader.load(ScreenLoader.class.getResource(viewPath));

        Scene view1Scene = new Scene(view1);
        stage.setScene(view1Scene);
        stage.show();
    }
}
```

Ahora, se puede llamar así:

```
@FXML
void goToView1(ActionEvent event) throws IOException {
    ScreenLoader.loadScreen("main.fxml", (Stage)
        ((Node) event.getSource()).getScene().getWindow());
}
```

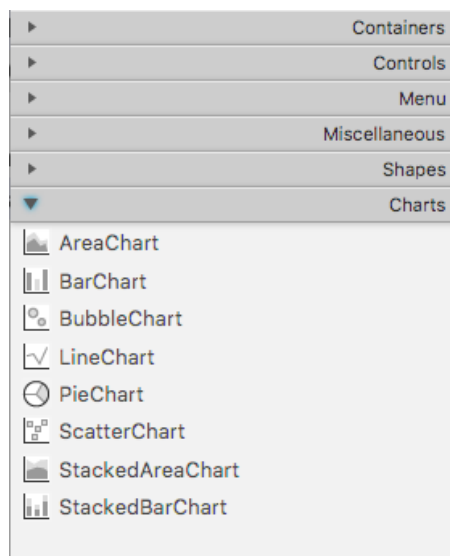
### Ejercicio 13

Completa el ejemplo anterior añadiendo los eventos para cambiar de vista en los botones que faltan.

## 13.6. Usando gráficos

Para terminar con esta sección, se verá cómo añadir otro tipo de control útil. Los gráficos (*charts*) permiten representar la información de una manera gráfica y resumida que a menudo es mucho más comprensible que leer datos de una tabla.

Puedes encontrar más tipos de gráficos en la lista de gráficos de *SceneBuilder*.

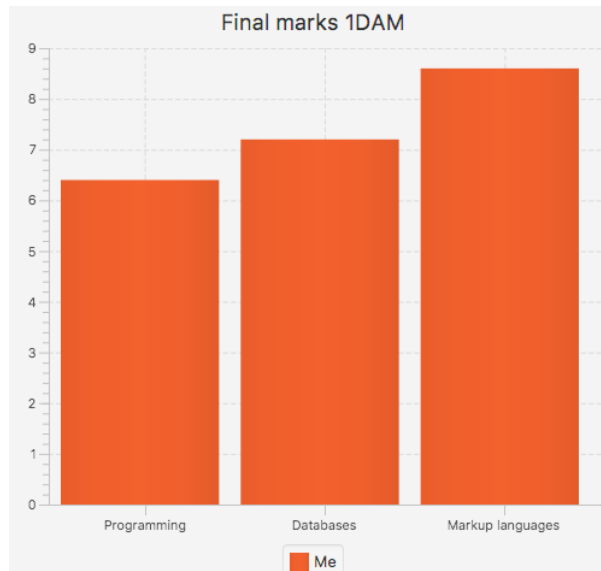


Para usar cualquiera de ellos, simplemente se arrastra a la escena, como con cualquier otro control. Dependiendo del tipo de gráfico, tendrá algunos atributos y métodos específicos para proporcionar los datos.

Puedes echar un vistazo a algunos tutoriales en el siguiente enlace.

<https://docs.oracle.com/javafx/2/charts/jfxpub-charts.htm>

Por ejemplo, si trabajas con un gráfico de barras (*BarChart*), necesitarás establecer el tipo de datos para el eje X (*categories*) y el eje Y (*values*). Si quieres representar algo como esto:



Entonces necesitarás añadir un elemento *BarChart* a la aplicación, y establecer un nombre apropiado para él (por ejemplo, *chart*). Después, en el controlador, necesitarás especificar el tipo de datos del eje X (*categories*) y el eje Y (*numbers*):

```
@FXML
private BarChart<String, Number> chart;
```

Se puede llenar el gráfico con algunos datos estáticos, como *arrays* predefinidos. Para crear cada barra, es necesario crear una instancia del objeto *XYChart.Data* y llenar un objeto *XYChart.Series* con esos datos.

```
String[] categories = {"Programming", "Databases", "Markup languages"};
double[] marks = {6.4, 7.2, 8.6};

chart.setTitle("Final marks 1DAM");

XYChart.Series data = new XYChart.Series();
data.setName("Me");

for (int i = 0; i < categories.length; i++) {
    data.getData().add(new XYChart.Data(categories[i], marks[i]));
}
chart.getData().add(data);
```

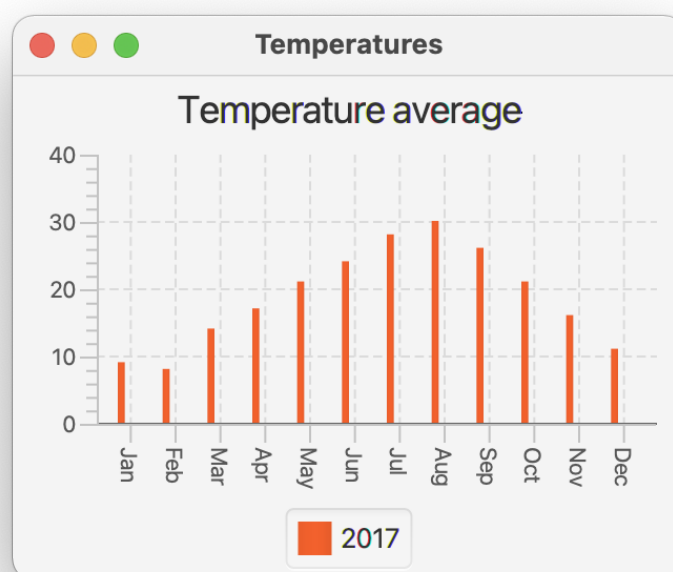
Como puedes ver, también se puede definir un título para el gráfico. También puedes verás que es posible añadir tantas series como se necesiten. Cada una estará representada con un color de barra diferente.

### Ejercicio 14

La siguiente tabla representa una lista de la temperatura media de cada mes en Alicante durante el año 2017.

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
9	8	14	17	21	24	28	30	26	21	16	11

Cree un nuevo proyecto JavaFX FXML llamado **Temperatures** y añada un gráfico de barras (control *BarChart*) para representar estos datos. Al final, debería verse algo como esto al iniciarse la aplicación:



## 14. Ejemplo: HealthyMenu

Para apoyar el aprendizaje de JavaFX y *SceneBuilder*, se creará una pequeña aplicación desde cero. Esta aplicación tendrá dos vistas: una para mostrar y editar los datos y otra para mostrar un gráfico que resuma los datos actuales. Para empezar, se creará un nuevo proyecto JavaFX FXML llamado **HealthyMenu**. Después, sigue los pasos que se dan en las siguientes secciones.

### 14.1. Diseño de la vista principal

La vista FXML principal tendrá la siguiente apariencia:

Food name	Food category	Weight (g)	Weight (oz)
Potatoes	Fruits and vegetables	200	7.05
Chicken	Meat and fish	300	10.58
Milkshake	Milk derivatives	250	8.82
Salmon	Meat and fish	300	10.58

Food name:

Food category:

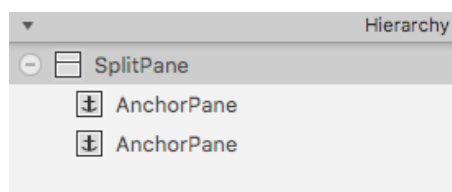
▼

Weight (g):

Add

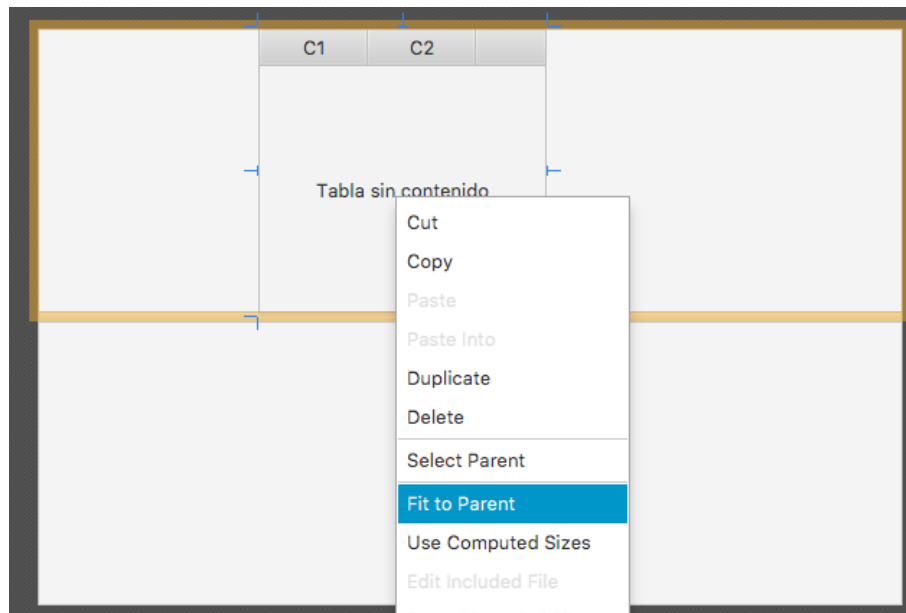
Chart >

Para obtener esta apariencia, puedes, por ejemplo, establece un *SplitPane* vertical como diseño principal. Por defecto, tendrás dos *AnchorPane* como *layouts* para sus dos mitades. Eso está bien para este ejemplo, aunque puedes cambiarlos de todos modos.



#### 14.1.1. Definiendo la tabla

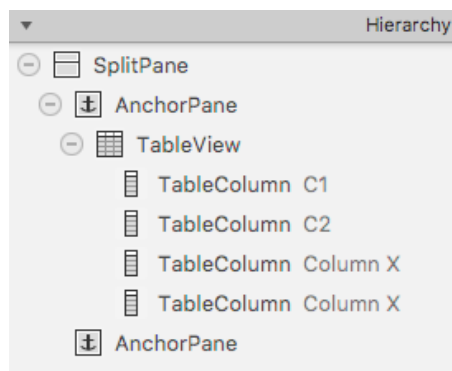
En la mitad superior del panel dividido, añade un *TableView*. Para que se ajuste al tamaño total del *AnchorPane*, haz clic derecho en la vista de tabla y elije la opción *Fit to Parent...*



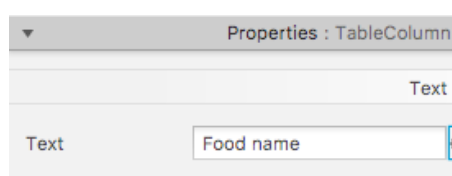
La tabla tendrá cuatro columnas:

- El nombre de un alimento que ingerido. Por ejemplo, *“Tomatoes”* o *“Milk”*.
- La categoría del alimento, que se elegirá entre un conjunto de valores predefinidos: *“Fruits and vegetables”*, *“Meat and fish”*, *“Milk derivatives”* y *“Other”*.
- La cantidad de comida en gramos.
- La cantidad de comida en onzas (oz), que se calculará automáticamente a partir de la columna anterior.

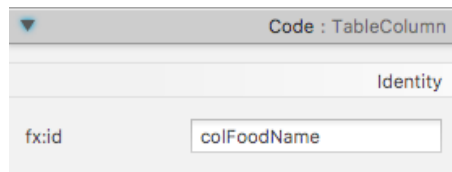
Para establecer el encabezado de estas cuatro columnas, solo se necesitará añadir tantos elementos *TableColumns* a la tabla como se necesiten:



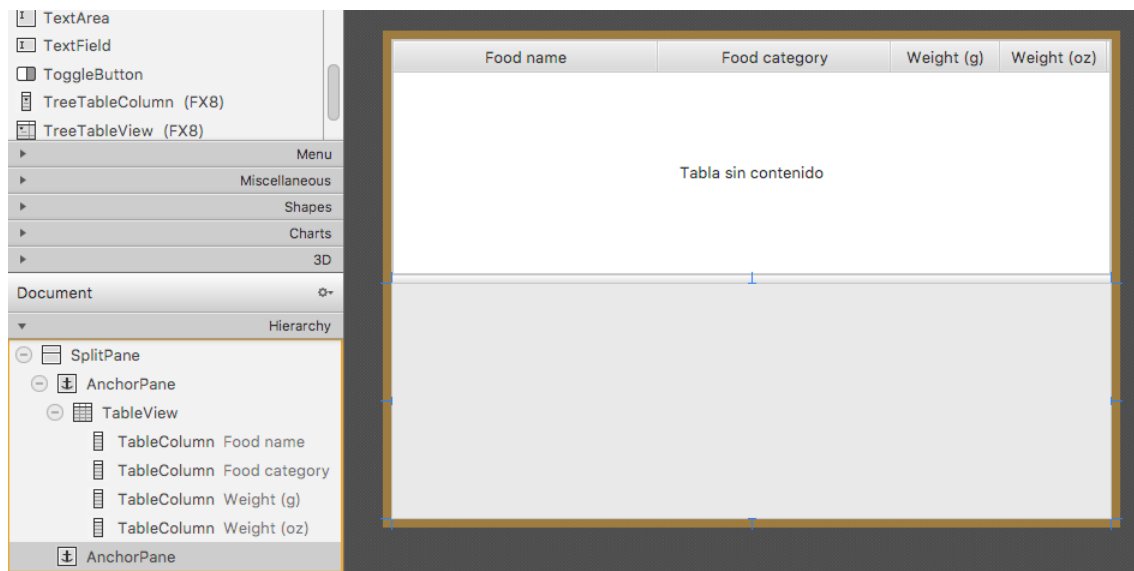
Luego, haz clic en cada *TableColumn* en el panel izquierdo y cambia sus propiedades en el panel derecho. Por ejemplo, para la primera columna, cambia su propiedad *Text* a *“Food name”*...



... y su propiedad `fx:id` para referenciarla desde el código:



Después de cambiar estas propiedades en las cuatro columnas, obtendrás esta tabla:

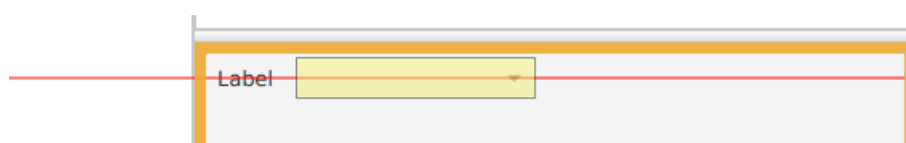


### 14.1.2. Definiendo la parte inferior

La parte inferior permitirá añadir comida a la tabla superior. Se necesitarán los siguientes campos de formulario:

- Un campo de texto para especificar el nombre del alimento (con su etiqueta asociada).
- Un cuadro de elección para elegir la categoría de alimentos (con su etiqueta asociada).
- Otro campo de texto para establecer el peso en gramos (con su etiqueta asociada).
- Un botón para agregar los datos a la tabla.

Se comenzará añadiendo cada par etiqueta y campo asociado. Para el primer par (*food name*), empieza por añadir la etiqueta y luego el campo de texto al lado. Como puedes ver cuando se mueve un elemento cerca de otros, o cerca de los bordes de un contenedor, aparecen líneas auxiliares que te ayudarán a alinear ese elemento con el mismo margen que otros, o alinear elementos.



Al terminar, deberías obtener algo como esto:

Para este ejemplo, se ha asociado el campo de texto del nombre del alimento, el campo de texto del peso, el cuadro de opciones y el botón con estos identificadores *fx:id* respectivamente: *txtFoodName*, *txtWeight*, *choiceCategory* y *btnAdd*. Asocia cada control un *fx:id* apropiado y actualiza el controlador.

Ahora, se llenará el cuadro de elección con la lista predefinida de posibles valores. En primer lugar, especifica el tipo de datos del desplegable:

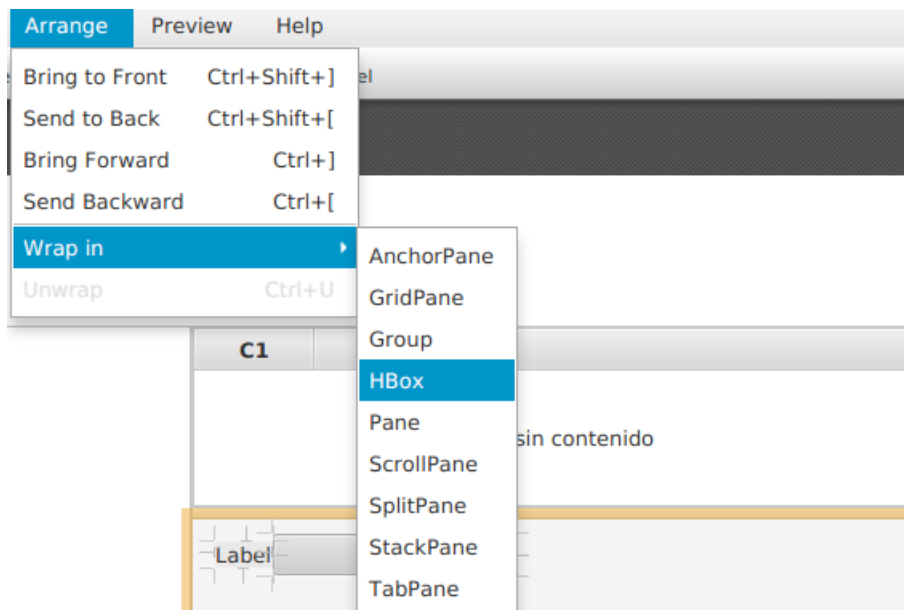
```
@FXML
private ChoiceBox<String> choiceCategory;
```

Luego, añade estas líneas al método *initialize()* del controlador principal:

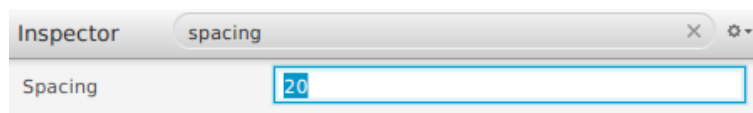
```
choiceCategory.setItems(
    FXCollections.observableArrayList(
        "Fruits and vegetables", "Meat and fish", "Milk derivatives", "Other"
    )
);
```

Para ayudar a que el formulario se adapte según se cambia el tamaño con la ventana, primero tendrás que decirle a los controles cómo cambiar el tamaño. Puede resultar útil envolver algunos controles en un *layout HBox* o *VBox*, para tratarlos como un todo. Por ejemplo, coloca la etiqueta *Food name*, su campo de texto asociado y el botón *Add* dentro de un *HBox*.

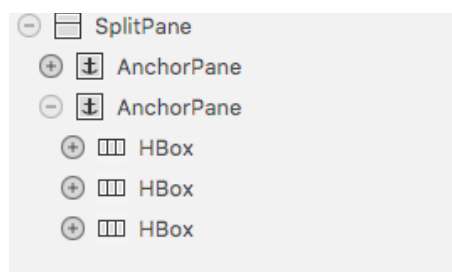
Para agruparlos dentro de un *HBox*, selecciona todos los controles involucrados (haz clic en ellos mientras presiona la tecla *Mayús*), y en el menú de aplicación de *SceneBuilder* selecciona **Arrange > Wrap in > HBox**.



Después, selecciona el espacio (margen) que separará los elementos unos de otros dentro del *HBox* (en píxeles).



Repite los mismos pasos para la filas *Food category* y *Weight*. Entonces, tendrás tres *layouts HBox* dentro del *AnchorPane*:

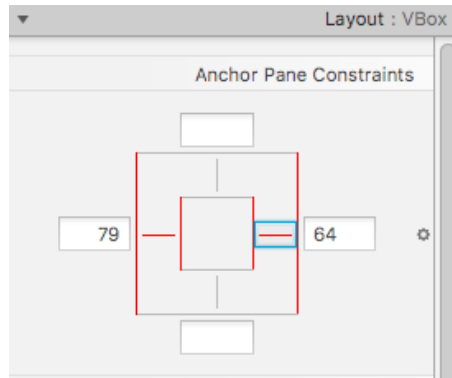


Ahora, podemos envolver estos *HBox* dentro de un *VBox* para tratar toda la forma a la vez. Selecciona los tres contenedores *HBox* y envuélvelos en un *VBox*, como se hizo antes para los controles de formulario. También podrás establecer el espaciado vertical para este *VBox*.



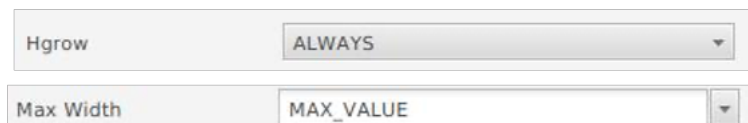


Finalmente, para permitir que el formulario crezca o se reduzca a medida que se cambie el tamaño de la ventana, selecciona el `VBox` y áncalo a sus bordes izquierdo y derecho:



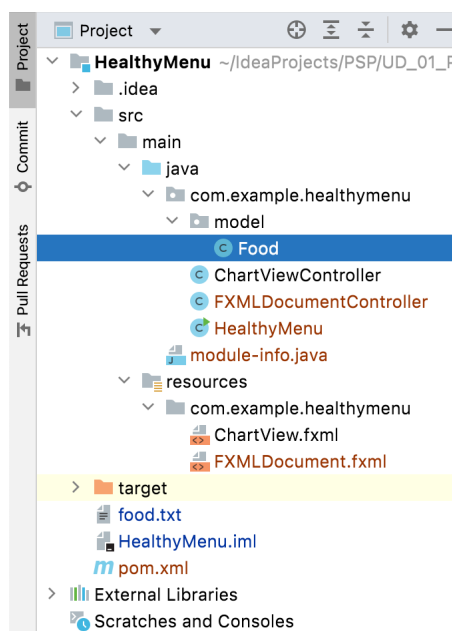
Si intentas cambiar el tamaño de la ventana ahora, notarás que las etiquetas pueden ocultar su contenido si se encogen demasiado. Para evitar esto, se puede establecer un ancho mínimo para ellas:

También puedes decirle a los campos de texto que crezcan tanto como sea posible si se aumenta el ancho de la ventana. Esto se puede configurar con las siguientes dos propiedades:



## 14.2. Cargar datos en la tabla

Es hora de llenar la tabla con algunos datos. Para hacerlo, se creará una clase llamada `Food` en un paquete llamado `com.example.healthymenu.model`.



Esta clase tendrá los siguientes atributos:

- *Food name* (*String*).
- *Food category* (*String*).
- *Food weight* en gramos (*int*).

Además, tendrá con un constructor para asignar estos valores, y los correspondientes *getters* y *setters*. Finalmente, se añadirá un *getter* adicional para calcular el peso en onzas a partir del atributo *weight*. Sobrecargar el método *toString()* también puede resultar útil para más adelante.

```
public class Food {
    String name;
    String category;
    int weight;

    public Food(String name, String category, int weight) {
        this.name = name;
        this.category = category;
        this.weight = weight;
    }

    public String getName() {
        return name;
    }

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

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public int getWeight() {
        return weight;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public float getWeightInOz() {
        return (float) weight * 0.035274f;
    }

    @Override
    public String toString() {
        return name + ";" + category + ";" + weight;
    }
}
```

Pasa ahora al controlador principal.

Para este ejemplo se ha llamado al objeto *TableView* como *tableFood*, y las columnas correspondientes son *colFoodName*, *colFoodCategory*, *colWeightG* y *colWeightOz*. Tendrás que especificar el tipo de datos de todos estos elementos, de modo que quedarán así:

```
@FXML
private TableView<Food> tableFood;
@FXML
private TableColumn<Food, String> colFoodName;
@FXML
private TableColumn<Food, String> colFoodCategory;
@FXML
private TableColumn<Food, Integer> colWeightG;
@FXML
private TableColumn<Food, Float> colWeightOz;
```

Se necesitará definir una colección de objetos *Food* para llenar la tabla. Como se aprendió en sesiones anteriores, las tablas utilizan un tipo especial de colección llamado *ObservableList* para rellenar los datos. Por tanto, crea un nuevo atributo de este tipo en el controlador:

```
ObservableList<Food> food;
```

Podemos inicializarlo con algunos datos de muestra en el método *initialize()*.

```
food = FXCollections.observableArrayList(
    new Food(" Potatoes", "Fruits and vegetables", 200),
    new Food(" Chicken", "Meat and fish", 300),
    new Food(" Milkshake", "Milk derivatives", 250),
    new Food(" Salmon", "Meat and fish", 300)
);
```

Finalmente, asocia esta lista a la tabla, estableciendo el atributo para cada columna (también en el inicializar método):

```
colFoodName.setCellValueFactory(new PropertyValueFactory("name"));
colFoodCategory.setCellValueFactory(new PropertyValueFactory("category"));
colWeightG.setCellValueFactory(new PropertyValueFactory("weight"));
colWeightOz.setCellValueFactory(new PropertyValueFactory("weightInOz"));

tableFood.setItems(food);
```

Tras estos pasos, deberías ver la tabla como la siguiente cuando lances la aplicación:

Food name	Food category	Weight (g)	Weight (oz)
Potatoes	Fruits and vegetables	200	7.0548
Chicken	Meat and fish	300	10.5822
Milkshake	Milk derivatives	250	8.8185
Salmon	Meat and fish	300	10.5822

Observa que se ha asociado cada columna de la tabla con un método *getter* de la clase *Food* por su nombre (sin el obtener prefijo). Aún se puede mejorar la apariencia de la tabla formateando el peso en onzas con 2 números decimales. Para ello, basta simplemente con devolver un *String* en el *getter* correspondiente (clase *Food*), en lugar de devolver solo un *float*:

```
public String getWeightInOz() {
    DecimalFormat df = new DecimalFormat();
    df.setMaximumFractionDigits(2);
    return df.format((float) weight * 0.035274f);
}
```

### 14.2.1. Leer/Guardar datos de/a un fichero

Para tener una aplicación más profesional, se reemplazará la lista de alimentos de muestra por una lista leída desde un archivo de texto. El archivo de texto tendrá la siguiente estructura:

```
Food name;Category;Weight in grams
```

Se puede crear un archivo llamado *food.txt* en la carpeta raíz del proyecto y rellenarlo con datos de muestra:

```
Potatoes;Fruits and vegetables;200
Chicken;Meat and fish;300
Milkshake;Milk derivatives;250
Salmon;Meat and fish;300
```

Después, se crearán dos métodos estáticos en el controlador (o en una clase separada, si planeas utilizarlos en otro lugar) para leer los datos del archivo (y devolver una lista de alimentos) y almacenar el datos, respectivamente.

```
private static List<Food> readFile() {
    try {
        return Files.lines(Paths.get("food.txt"))
            .map(line -> new Food(line.split(";")[0],
                line.split(";")[1], Integer.parseInt(line.split(";")[2])))
            .collect(Collectors.toList());
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

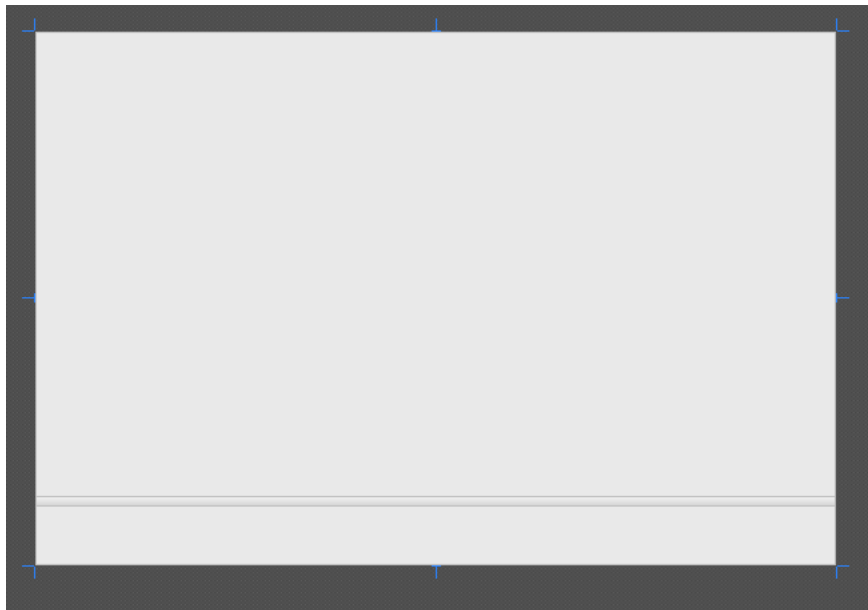
private static void saveFile(List<Food> food) {
    try (PrintWriter pw = new PrintWriter("food.txt")) {
        food.stream().forEach(f -> pw.println(f.toString()));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ten en cuenta que se utiliza el método *toString()* para guardar en el archivo la salida generada automáticamente por la nueva versión del método de la clase *Food*. Observa lo fácil que es leer, o escribir, en un fichero de texto utilizando la programación funcional (*streams* y *lambdas*).

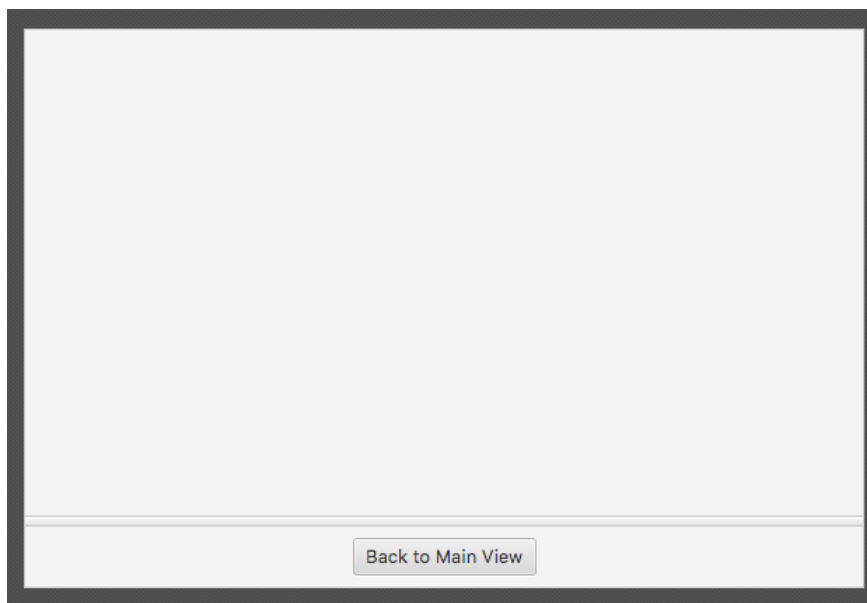
### 14.3. La vista del gráfico

Crea ahora la vista secundaria. Será una vista con un gráfico circular y un botón para volver a la vista principal. Añade un archivo FXML vacío dentro de *resources*, dentro de la carpeta *com.example.healthymenu*. Puedes llamarlo *ChartView.fxml*. Ahora, ve al paquete *com.example.healthymenu* (dentro de la carpeta Java) y añade un nuevo controlador (*ChartViewController*) asociado a la nueva vista FXML, recuerda que puede hacerse desde *SceneBuilder*.

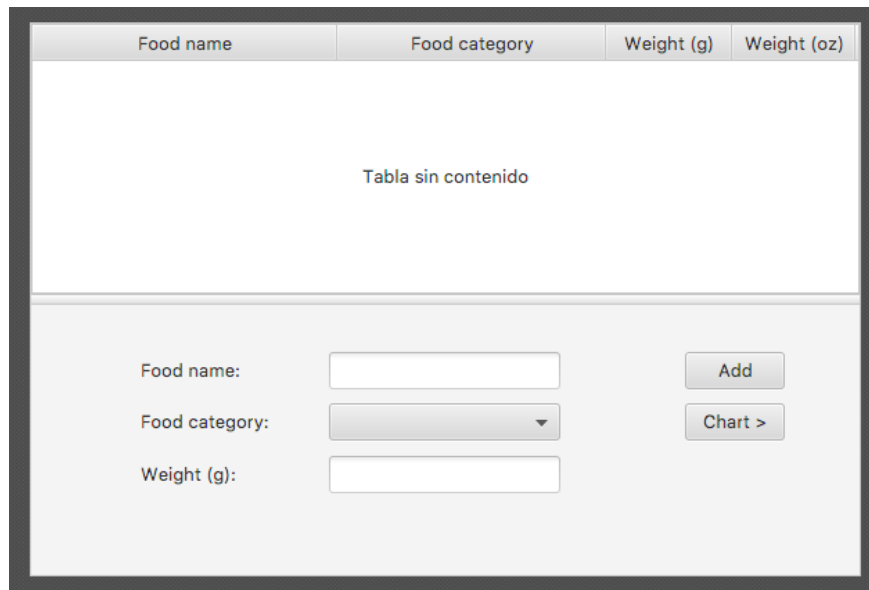
Después, edita el archivo FXML con *SceneBuilder*. Puedes utilizar un *layout SplitPane* vertical para el diseño principal, con el panel superior más grande que el panel inferior:



Añade un Gráfico circular en el panel superior (y ajústalo al *parent*), y un botón en medio del panel inferior. Deberías obtener algo como esto:



Para mostrar esta vista desde la principal, se añadirá un nuevo botón en esa vista principal. Puedes hacerlo bajo el botón *Add*:



Food name	Food category	Weight (g)	Weight (oz)
Tabla sin contenido			

Food name:

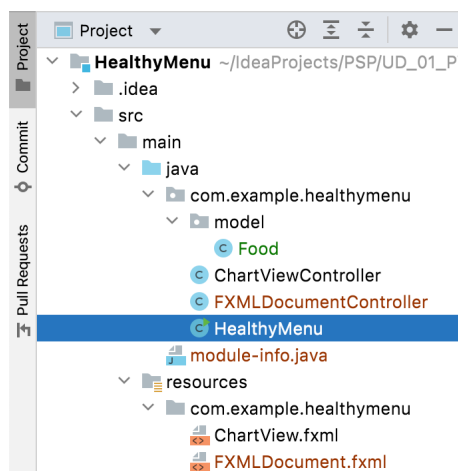
Food category:

Weight (g):

Add Chart >

## 14.4. Modularidad

Es necesario definir el fichero *module-info.java* para poder utilizar la siguiente estructura de paquete Java:



Por suerte, desde la última versión de, ya no es necesario preocuparse por este fichero, pero sí conocerlo. *Exports* indica que el paquete principal puede acceder al paquete *com.example.healthymenu.model*. Además, la cláusula *opens* indica que un paquete debe ser mostrado. Revisa que tu fichero *module-info.java* sea algo parecido:

```
module com.example.healthymenu {
    requires javafx.controls;
    requires javafx.fxml;

    exports com.example.healthymenu;
    exports com.example.healthymenu.model;

    opens com.example.healthymenu to javafx.fxml;
}
```

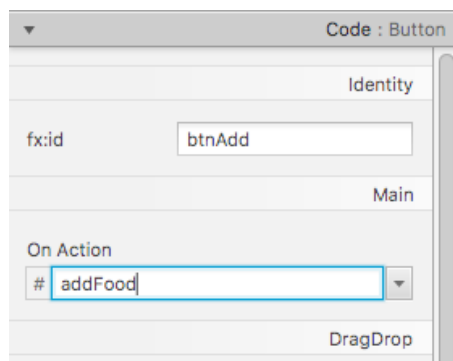
## 14.5. Añadir eventos

Para finalizar el comportamiento básico de la aplicación, necesitas añadir estos tres eventos:

- Un evento para añadir un nuevo alimento a la tabla cada vez que se haga clic en *Add* y luego guarde los cambios en el archivo.
- Un evento para cambiar a la vista del gráfico y actualizar su contenido cada vez que se haga clic en el botón *Chart>*.
- Un evento para volver a la vista principal cuando se haga clic en el botón *Back to Main View*.

### 14.5.1. Añadir un nuevo alimento a la tabla

Para añadir un nuevo alimento cuando se haga clic en el botón *Add*, primero se especificará el evento *addFood()* que se disparará cuando se pulse el botón:



El método en el controlador de la vista principal quedará como se muestra:

```
@FXML
private void addFood(ActionEvent event) {
    if (txtName.getText().equals("") ||
        choiceCategory.getSelectionModel().getSelectedIndex() < 0 ||
        txtWeight.getText().equals("")) {

        Alert dialog = new Alert(Alert.AlertType.ERROR);
        dialog.setTitle("Error");
        dialog.setHeaderText("Error adding data");
        dialog.setContentText("No field can be empty");
        dialog.showAndWait();
    } else {

        food.add(
            new Food(
                txtName.getText(),
                choiceCategory.getSelectionModel().getSelectedItem(),
                Integer.parseInt(txtWeight.getText())
            )
        );
        saveFile(food);
    }
}
```

Como puedes ver, se verifica si hay algún campo vacío. Si no, se añade un nuevo objeto *Food* a la lista (se supondrá que el usuario especificará los datos correctos en cada campo).

### 14.5.2. Cambio a la vista gráfico

Siempre que quieras ver la vista del gráfico, deberás pasar a esta vista la lista de alimentos actual. Por tanto, es buena idea definir un *getter* público para este atributo en el controlador principal:

```
public List<Food> getFood() {  
    return food;  
}
```

Después, define un nuevo evento en el botón *Chart* para cambiar a la vista del gráfico:

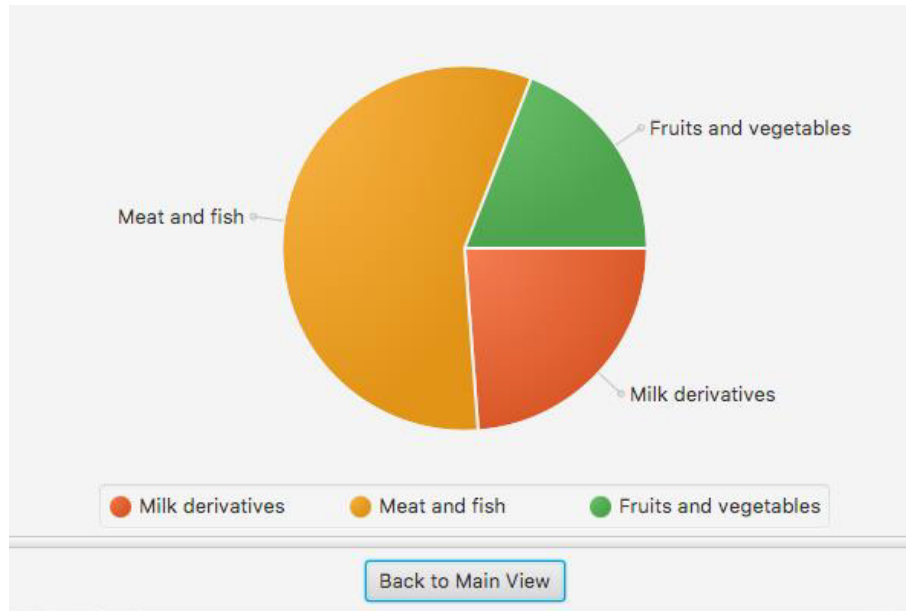
```
@FXML  
private void goToChartView(ActionEvent event) throws IOException {  
    FXMLLoader loader = new FXMLLoader(getClass().getResource("ChartView.fxml"));  
    Parent view1 = loader.load();  
    Scene view1Scene = new Scene(view1);  
  
    Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();  
    stage.hide();  
    stage.setScene(view1Scene);  
    stage.show();  
}
```

Dentro del controlador de la vista del gráfico, en el método *initialize()* define el código para obtener la lista de alimentos del controlador principal y, a continuación, actualizar los datos del gráfico circular.

```
public void initialize() {  
    // Get the controller object to get to the food list  
    FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLDocument.fxml"));  
  
    try {  
        Parent root = (Parent) loader.load();  
    } catch (Exception e) {}  
  
    FXMLDocumentController controller = (FXMLDocumentController) loader.getController();  
  
    List<Food> food = controller.getFood();  
  
    // Update the pie chart data from the food list data  
    pieChart.getData().clear();  
  
    // We get a map with all the categories and the sum of its weights  
    Map<String, Integer> result;  
  
    result = food.stream()  
        .collect(Collectors.groupingBy(  
            f -> f.getCategory(),  
            Collectors.summingInt(f -> f.getWeight())  
        ));  
    // We add an entry for the pie chart with each category and its sum  
    result.forEach((cat, sum) -> {  
        pieChart.getData().add(new PieChart.Data(cat, sum));  
    });  
}
```



Observa cómo se alcanza el controlador principal, se obtiene la lista de alimentos y luego (nuevamente) se utiliza la programación funcional para crear fácilmente un mapa con las categorías y la suma de sus pesos correspondientes. Esto es lo que se verá reflejado en el gráfico circular.



### 14.5.3. Cambio hacia la vista principal

Para finalizar, es necesario añadir un evento al botón *Back to Main View* en el *ChartViewController* para volver a la vista principal.

```
@FXML
private void back(ActionEvent event) throws IOException {
    FXMLLoader loader = new FXMLLoader(this.getClass().getResource("FXMLDocument.fxml"));
    Parent view1 = loader.load();
    Scene view1Scene = new Scene(view1);
    Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();

    stage.hide();
    stage.setScene(view1Scene);
    stage.show();
}
```

**NOTA:** también se podría haber usado la clase *ScreenLoader* comentada en secciones anteriores para evitar duplicar el código para cambiar de vista.

**NOTA:** cada vez que se cambie de vista en una aplicación, el correspondiente método *initialize()* del controlador, será llamado. Tenlo en mente.

### Ejercicio 15

Actualiza el proyecto *HealthyMenu* añadiendo un botón “Delete” en la parte inferior del formulario, con esta apariencia:

The screenshot displays a web application interface for 'HealthyMenu'. At the top, there is a table with four columns: 'Food name', 'Food category', 'Weight (g)', and 'Weight (oz)'. The table body is empty, showing the text 'Tabla sin contenido'. Below the table is a form with three input fields: 'Food name:' (a text box), 'Food category:' (a dropdown menu), and 'Weight (g):' (a text box). To the right of these fields are three buttons: 'Add' (a light gray button), 'Chart >' (a light gray button), and 'Delete' (a red button). The 'Delete' button is positioned below the 'Chart >' button.

Cuando se haga clic en ese botón, se eliminará la fila actualmente seleccionada de la tabla (y también de la lista asociada y del archivo de texto). Antes de eliminar la fila, deberá aparecer un cuadro de diálogo de confirmación, preguntando al usuario si realmente quiere eliminar la fila seleccionada.