Uso de layouts

El objetivo de este documento es explicar los diferentes layouts para organizar los componentes de una aplicación JavaFX

Por cada layout, se adjuntará un ejemplo que contiene un paquete con un archivo FXML y una clase main para hacer pruebas. El código de la clase main tendrá la siguiente estructura.

```
private (ClaseLayout) rootLayout;

@Override
public void start(Stage primaryStage) {
    try {
      // Carga el diseño del archivo FXML en la variable rootLayout
      FXMLLoader loader = new FXMLLoader();
      loader.setLocation(Main.class.getResource("Nombre Layout.fxml"));
      rootLayout = (ClaseLayout) loader.load();

      // Mostramos la escena del BorderPane de la variable rootLayot
      Scene scene = new Scene(rootLayout);
      primaryStage.setScene(scene);
      primaryStage.show();
   } catch(Exception e) {
       e.printStackTrace();
   }
}

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

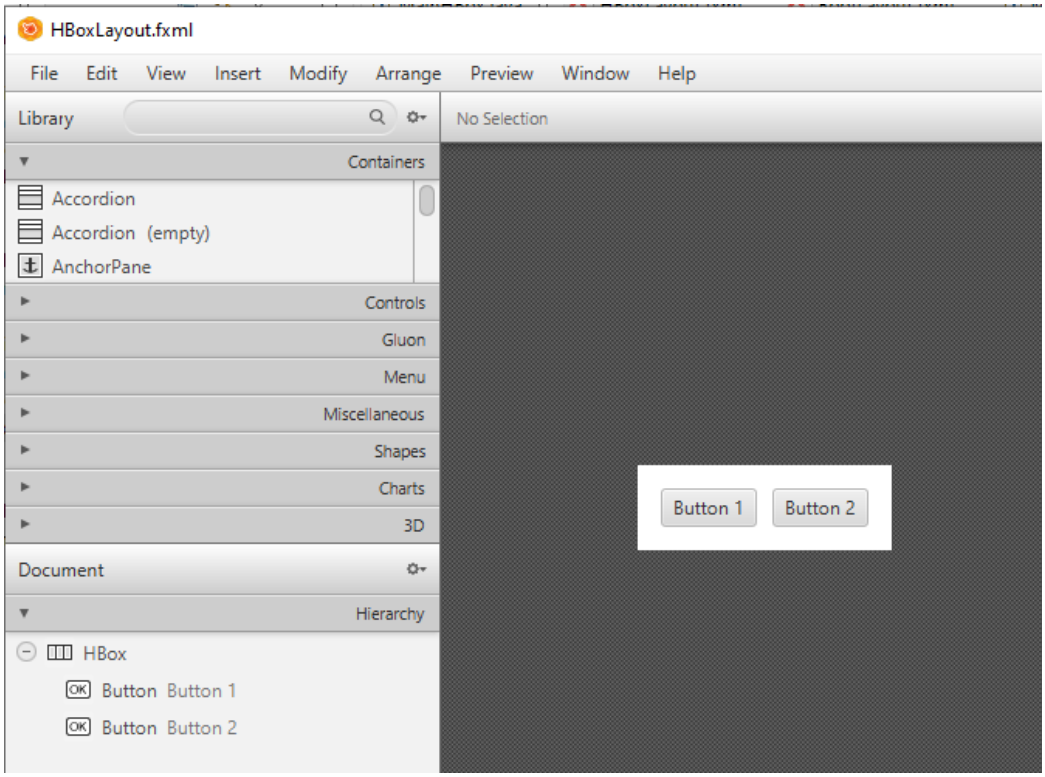
Como se puede observar en el código de arriba, crearemos una propiedad de la clase correspondiente al layout de cada apartado y se enlazará el FXML asociado al diseño.

1. HBox

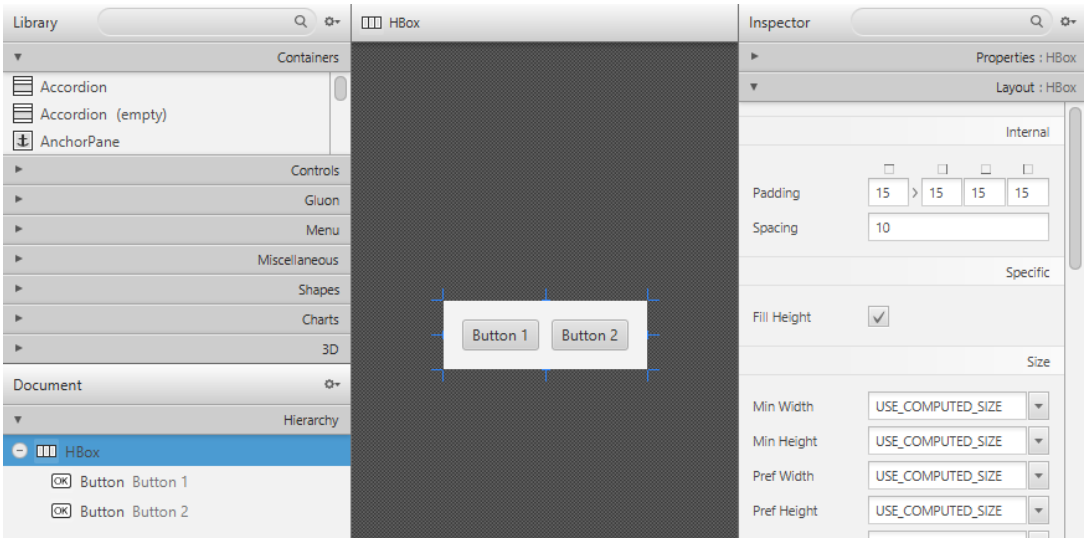
Este layout nos permite colocar los componentes hijos en una distribución en forma de fila alineados de izquierda a derecha.

Para ello, crearemos un archivo FXML con un **HBox** como "Root element" (elemento raíz) para probar su funcionamiento.

Abrimos el SceneBuilder y se arrastran dos botones al layout **HBox** como se muestra debajo.

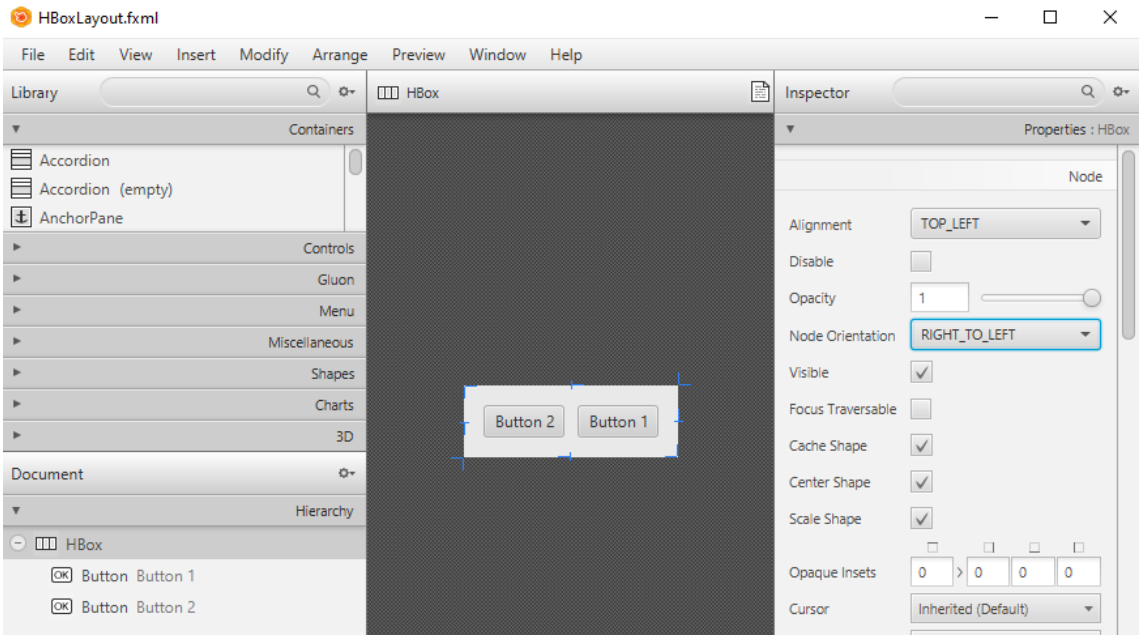


Además, añadiremos un **espaciado** entre elementos y **padding** entre el **HBox** y los botones. Esto se puede modificar desde la pestaña Layout con los valores que se indican a continuación.

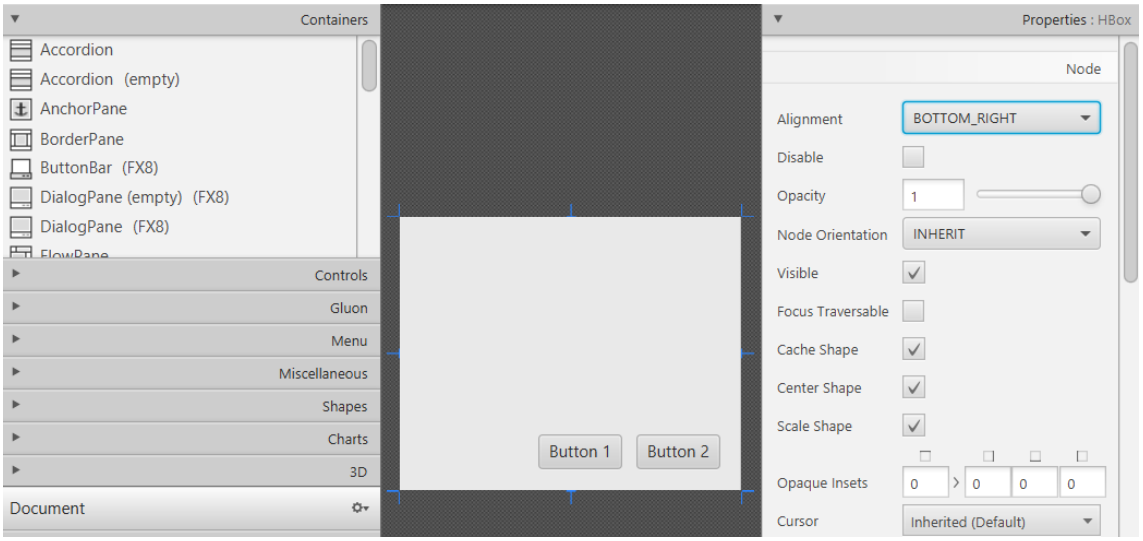


Al iniciar la aplicación, podrás observar que todos los elementos se sitúan de izquierda a derecha.

También existe la posibilidad de cambiar la orientación en el apartado Properties con la característica **Node Orientation**, entre otras muchas posibilidades.



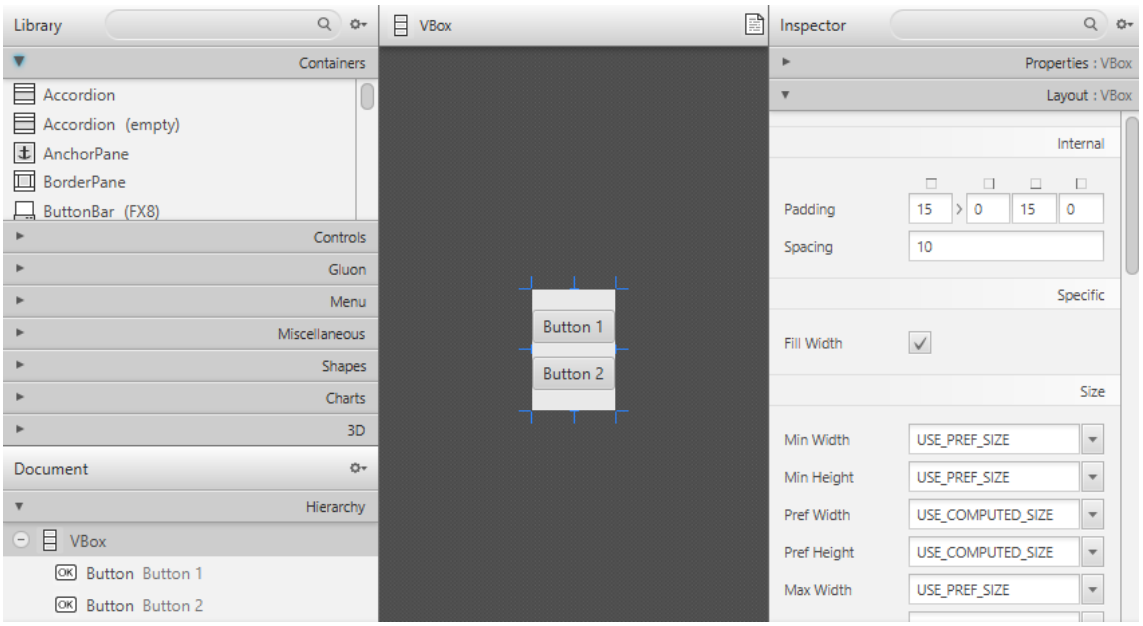
Por último, si asignamos un tamaño con **Pref Width** y **Pref Height** dentro de la pestaña "Layout", es posible modificar la alineación de los elementos mediante la propiedad **Alignment**. En el ejemplo que se muestra a continuación se ha seleccionado debajo a la derecha.



2. VBox

El funcionamiento de **VBox** es exactamente igual a HBox, pero en orientación de columna.

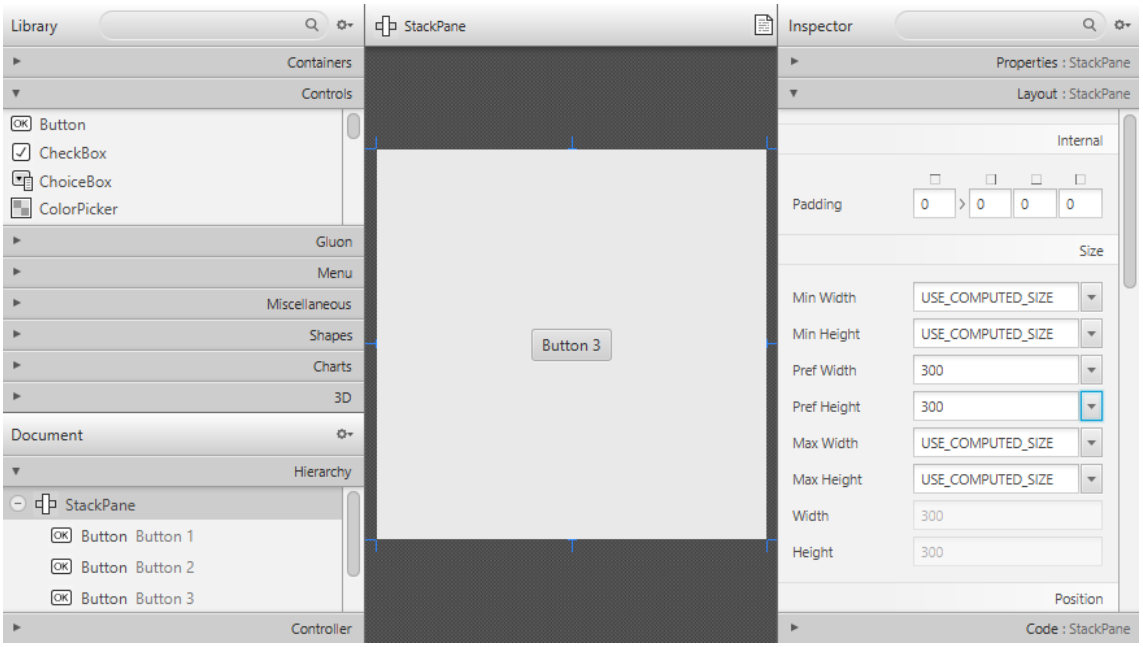
Por tanto, el ejemplo siguiente se configura de la misma manera, pero cambiando el layout a **VBox** a la hora de crear el FXML y enlazarlo en la clase Main.



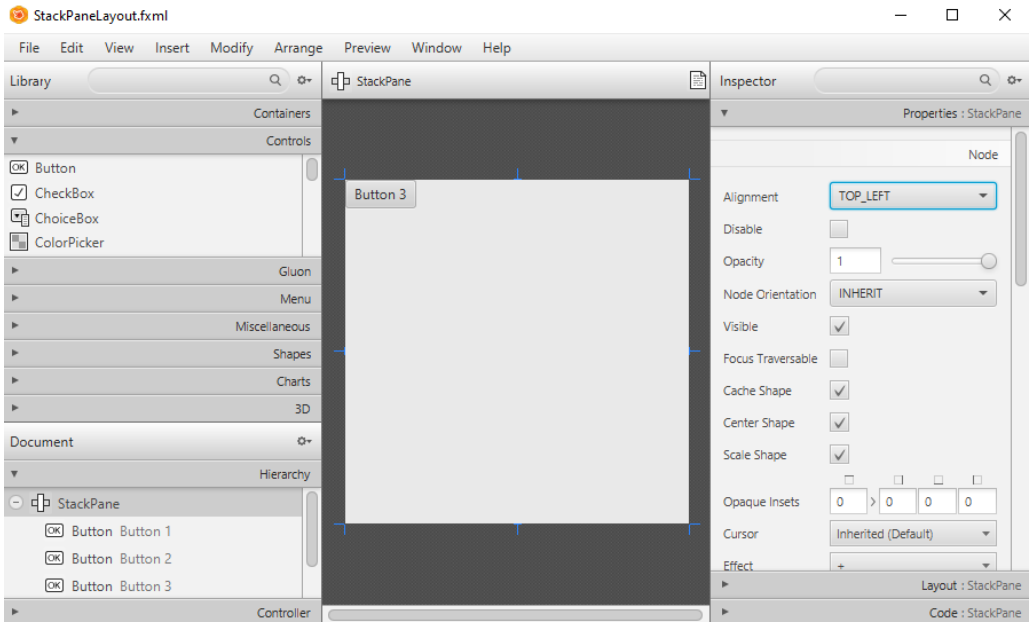
3. StackPane

Se trata de un layout un tanto peculiar, puesto que este agrega sus nodos en una pila, el primer elemento queda debajo del segundo y así sucesivamente.

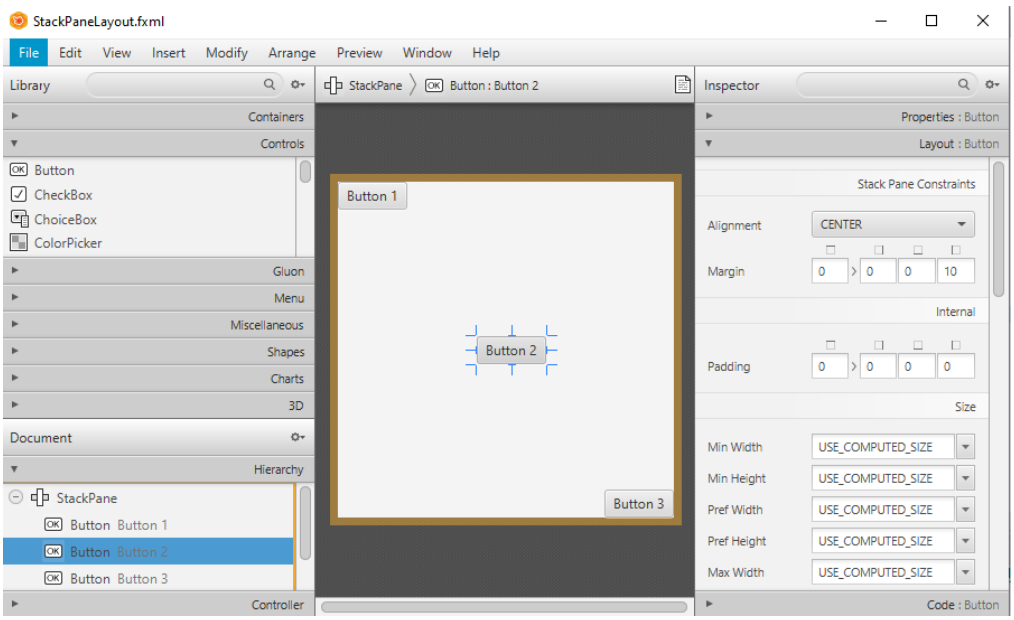
Una vez más probaremos con botones. Si añadimos tres al **StackPane**, obtendremos un resultado que no parece tener demasiada utilidad, ya que los botones aparecen superpuestos entre sí aun ampliando las dimensiones del layout.



Incluso si modificamos la propiedad **Alignment** del StackPane, se desplazarán todos los botones.



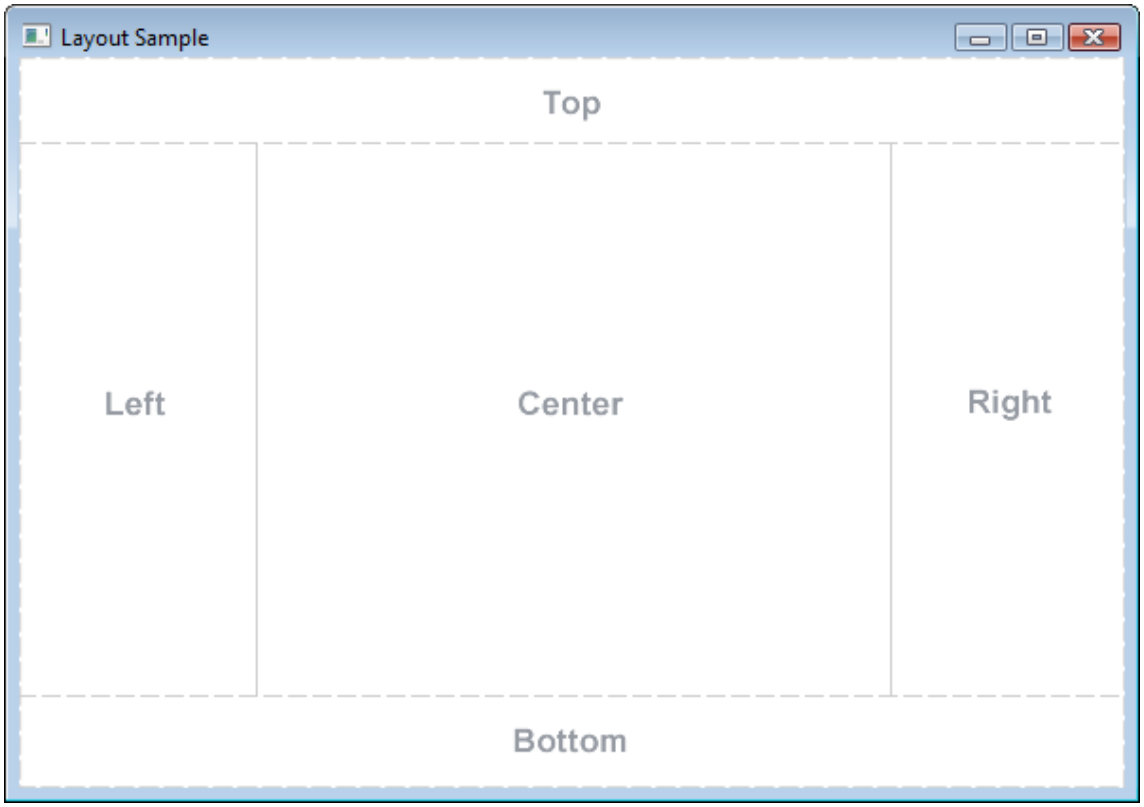
Ahora bien, si modificamos la alineación o desplazamos los botones con **Margin** u otra propiedad, entonces cambiará la visualización.



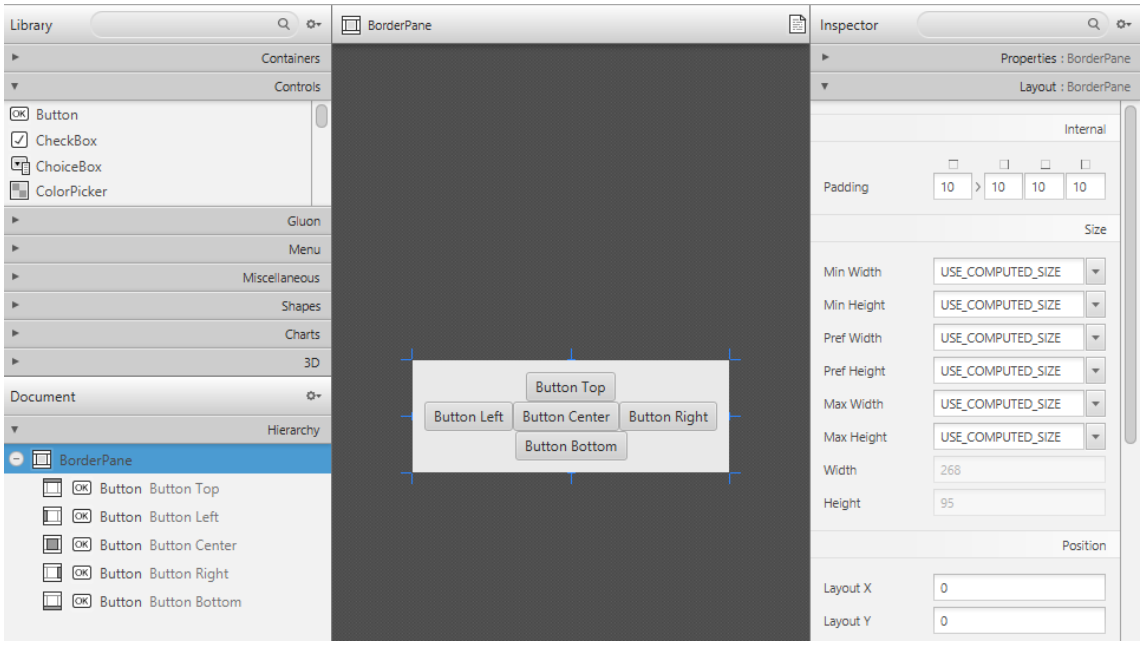
La mayor utilidad de este layout es crear formas complejas superponiendo diferentes elementos, como texto sobre una imagen o color de fondo. Por ejemplo, en la captura que se muestra debajo se combinan un elemento de tipo “Rectangle” y otro de tipo “Text” sin necesidad de incluir una imagen externa con un efecto similar.

4. BorderPane

Este layout permite agregar 5 nodos hijos en distintas posiciones como se muestra en la imagen: top, left, right, bottom y center.

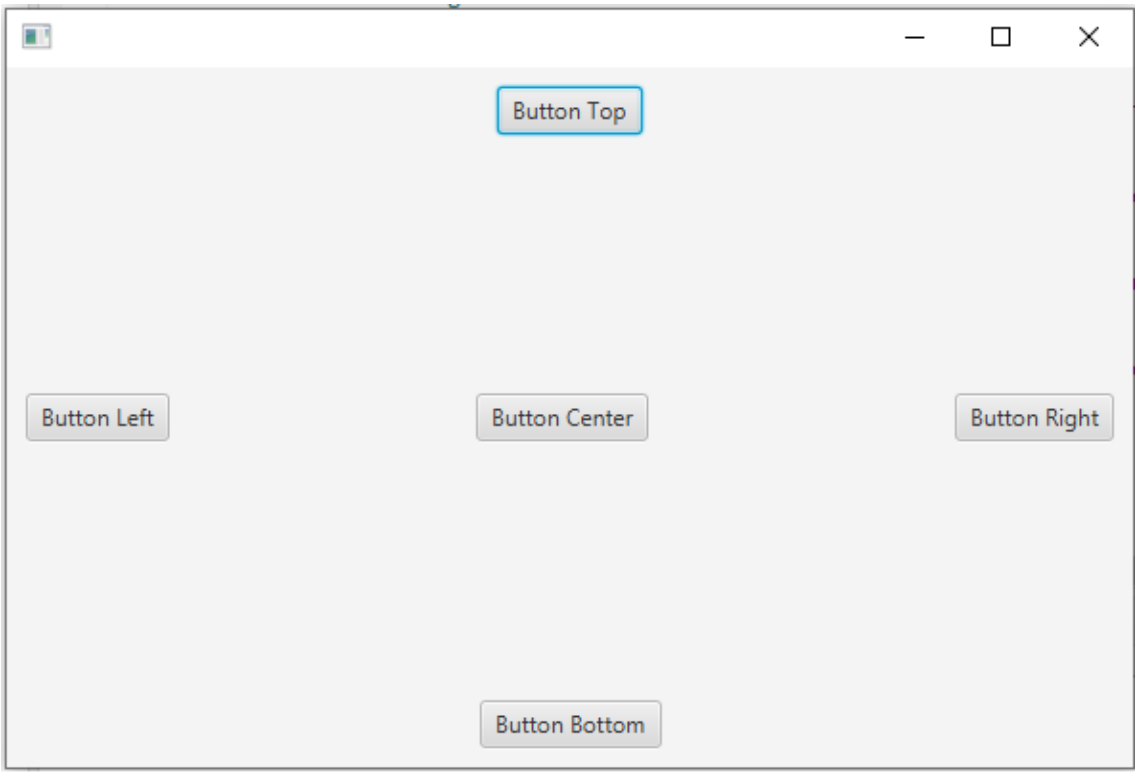


Para probar este layout, vamos a arrastrar un botón a cada una de las regiones obteniendo el siguiente resultado con un padding de 10 en el **BorderPane**:

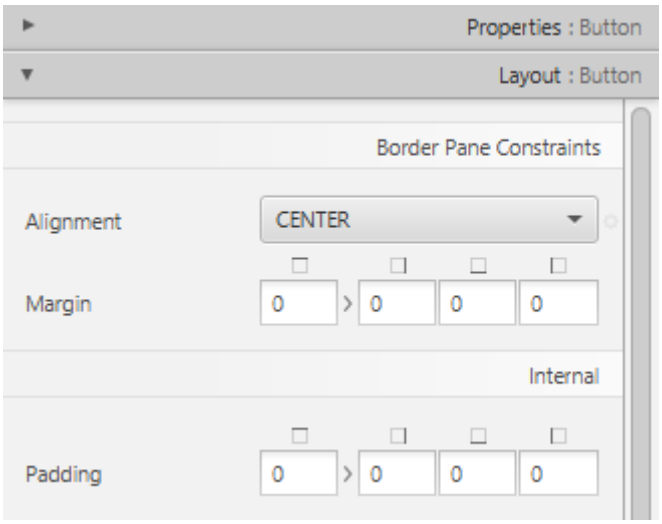


Para añadir espaciado entre las diferentes regiones habría que modificar la propiedad **Margin** en cada una de ellas.

Si agrandamos la ventana veremos el siguiente resultado:



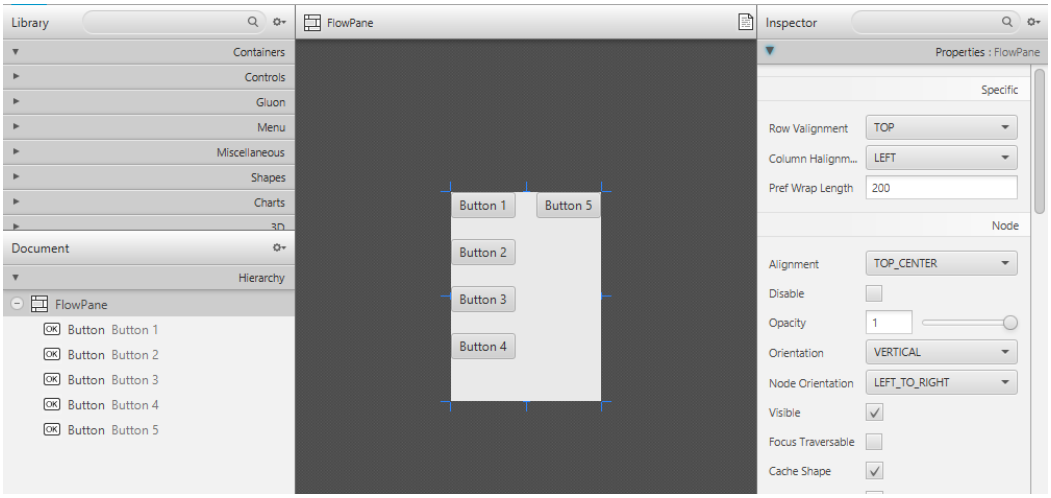
El comportamiento por defecto centra cada una de las regiones, pero se puede cambiar en la pestaña **Layout**, donde también se modifica el espaciado entre elementos que se indicaba en Margin.



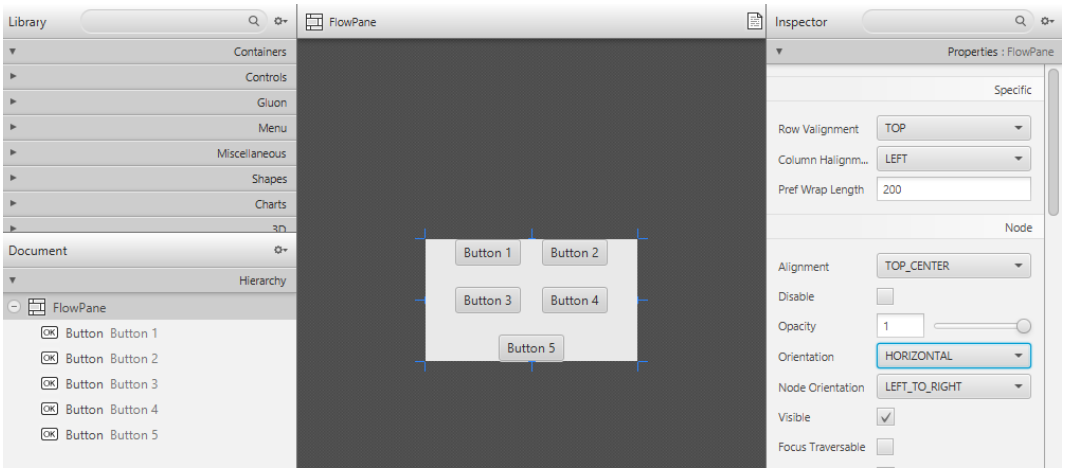
5. FlowPane

Este tipo de layout es parecido al HBox y al VBox. Puede ser como cualquiera de los dos según el orden de los elementos que establezcamos.

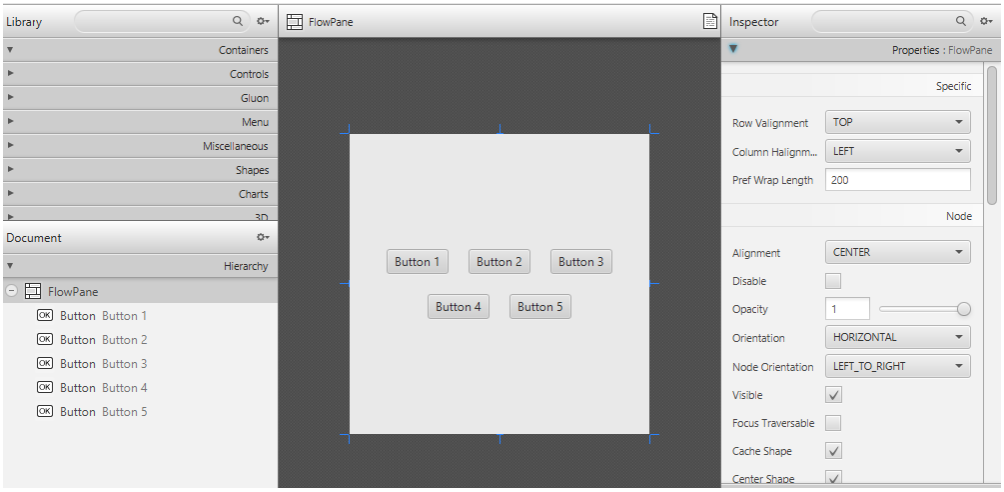
De hecho, vamos a emplear el mismo ejemplo, aunque en este caso arrastrando más botones que tendrán la siguiente disposición:



Si echamos un vistazo a la pantalla anterior, la propiedad **ORIENTATION** tiene el valor VERTICAL que hace que los nodos se sitúen de arriba a abajo dentro del ancho disponible como si fuese un VBox. Si cambiamos a HORIZONTAL, entonces el comportamiento será el de un HBox.



Cabe destacar, que se ha establecido la propiedad **Pref Wrap Length** que especifica un ancho o alto mínimo siempre que no lo especifiquemos en las propiedades **Pref Width** o **Pref Height** de la pestaña layout. Por ejemplo, en la imagen de debajo estos dos valores sobrescriben a Pref Wrap Length



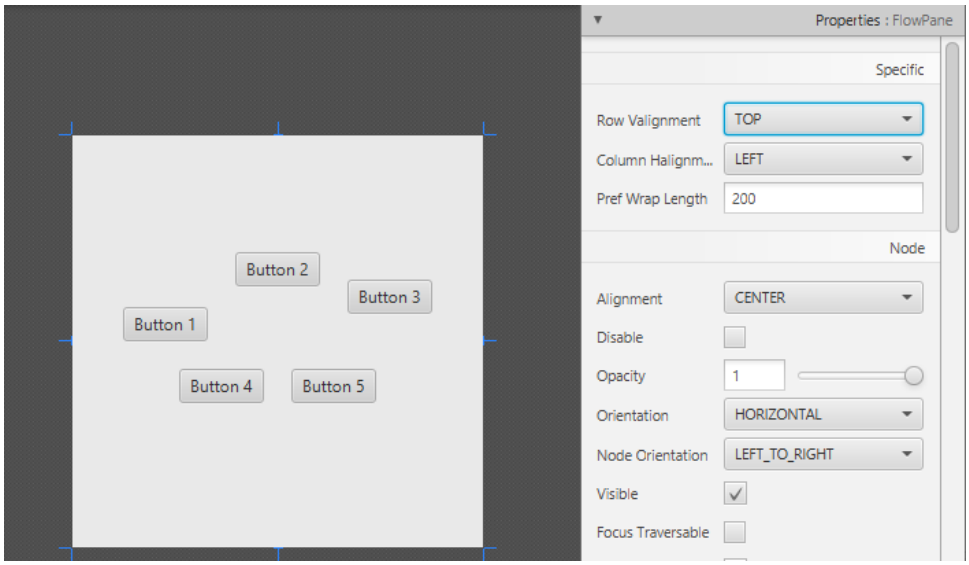
Por otro lado, en la misma imagen sobre este párrafo se puede observar que la propiedad **Alignment** tiene el valor CENTER. Esta propiedad nos permite todas las combinaciones posibles para alinear vertical y horizontalmente los ítems del layout.

Cabe no confundir con **Row Valignment** y **Column Haligmnent**. Estas últimas propiedades solamente son visibles cuando la altura de cada "fila" o "columna" de elementos tiene un tamaño diferente.

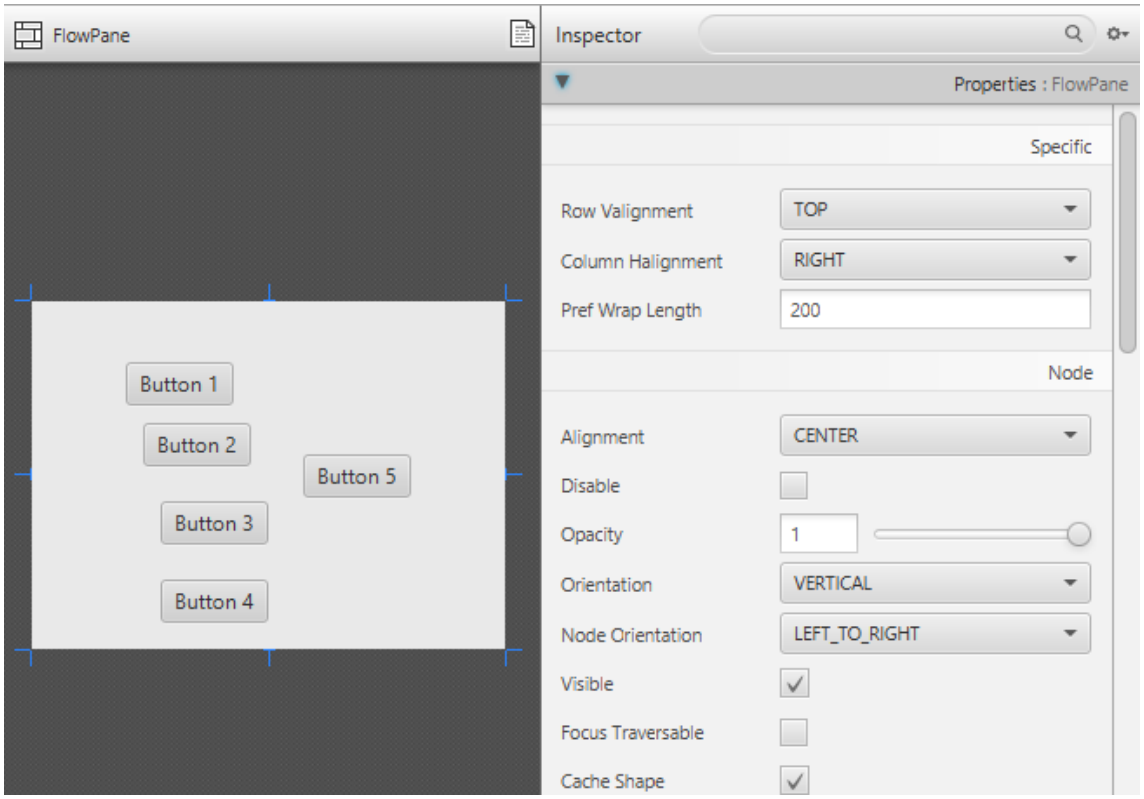
Es decir, vamos a establecer márgenes con la propiedad **Margin** de los botones dentro de la pestaña **Layout** para que las alturas sean diferentes.

Si la orientación es horizontal, solamente funciona **Row Valignment**. Se crean una especie de líneas imaginarias sobre cada fila de elementos y se refiere a la posición dentro de este espacio.

Por ejemplo, en la imagen de debajo se sitúan los nodos lo más arriba posible, pero prueba a cambiar las diferentes propiedades. La más compleja podría ser **BASELINE** que realiza los cálculos correspondientes para situar todos los elementos a la misma altura y a veces puede ocurrir que se desborde (cuidado con no asignar el valor BASELINE cuando la orientación es vertical, ya que no se debería aplicar, pero hay un error en JavaFX que descuadra el diseño).

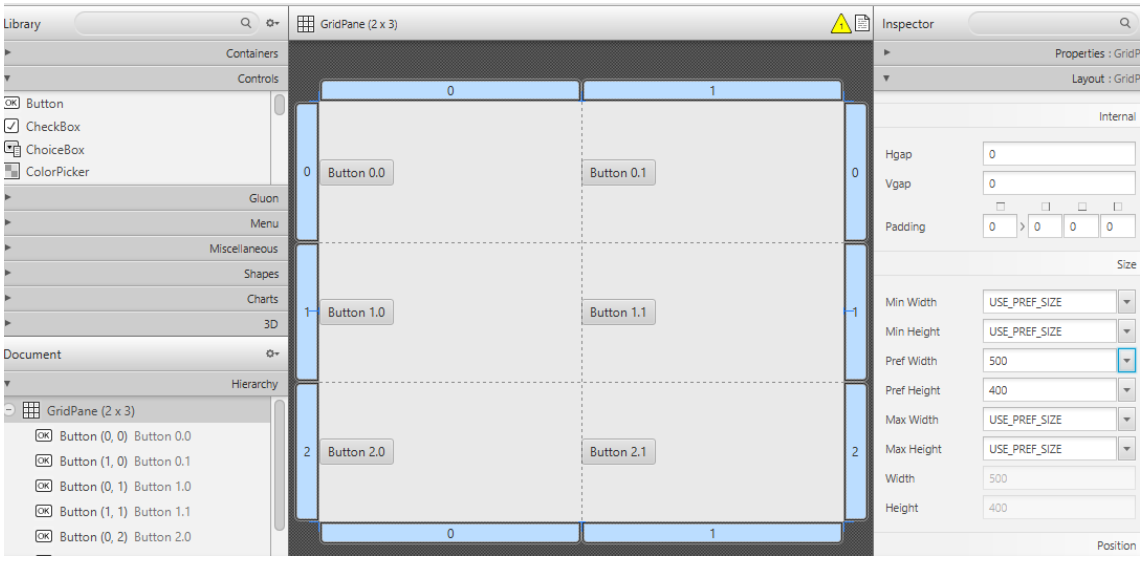


Por otro lado, si la orientación es vertical, solamente funciona la propiedad **Column Haligmnent.** En la imagen de debajo se establece a **RIGHT**, pero también admite alineación a la izquierda y centrado sobre la línea imaginaria en cada columna mencionada anteriormente.



6. GridPane

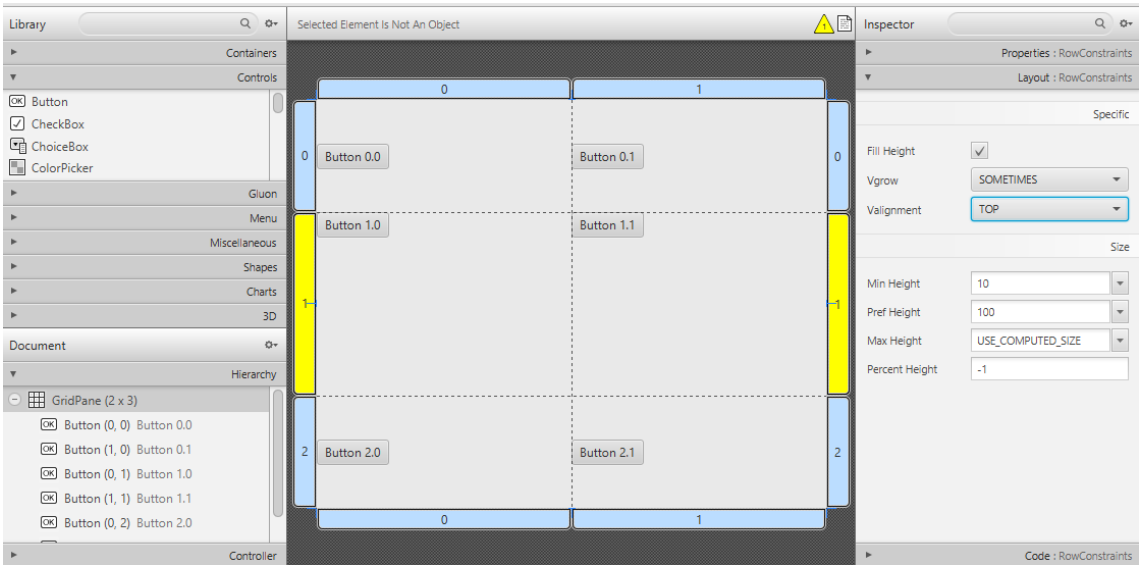
Permite añadir nodos en filas y columnas como si se tratara de una matriz o cuadrícula. SceneBuilder crea por defecto una cuadrícula de 3x2 con un tamaño predefinido en la que añadiremos botones en cada celda.



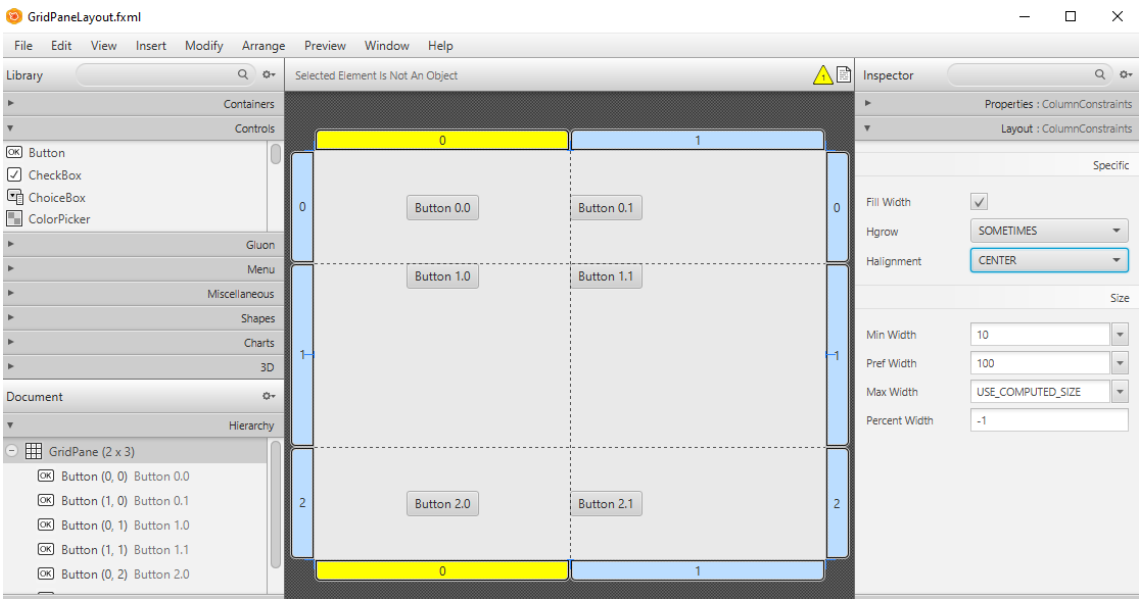
El tamaño de la pestaña Layout de la imagen se puede modificar y son las dimensiones generales del Grid.

Al margen de este alto y ancho, también es posible redimensionar cada una de las filas y columnas situando el cursor sobre los botones azules que indican la numeración.

Por ejemplo, vamos a modificar el ancho de la fila etiquetada como 1. Al modificar este alto, el resto de filas se redimensionan para cumplir la altura general del GridPane. Además, también es posible cambiar la alineación vertical mediante **Valignment** como se muestra debajo.

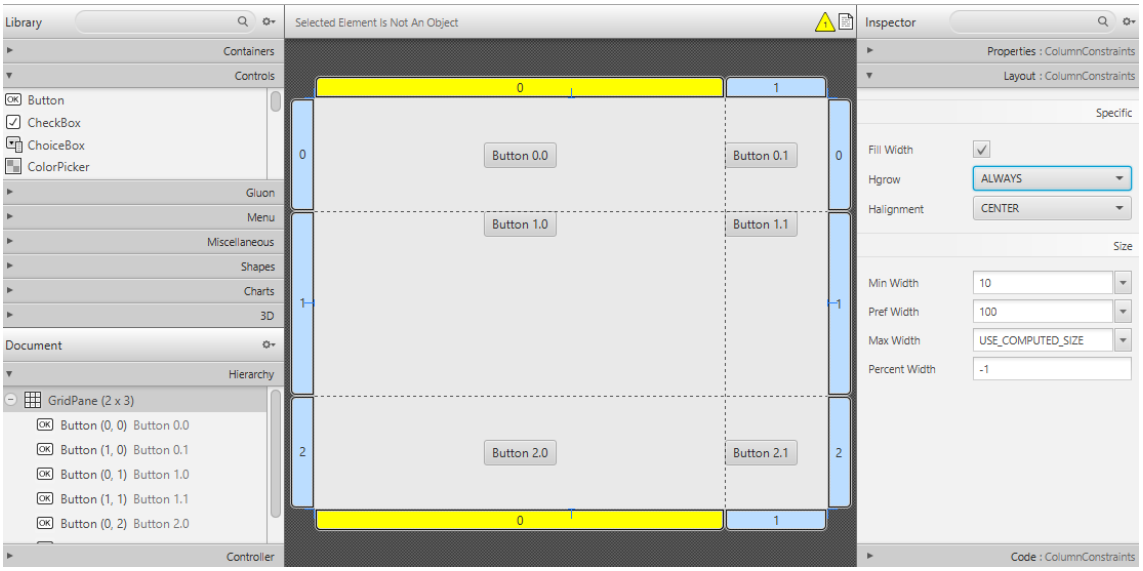


La alineación horizontal de los componentes a nivel de fila se hará en cada etiqueta. En este caso sería a través de **Halignment** del propio botón. Aunque también podemos cambiar la alineación horizontal a nivel de columna situándonos por encima de ellas y modificando **Halignment**



Otra propiedad interesante es **Hgrow** o **Vgrow** (en función de si estamos en una fila o columna). Por ejemplo, en el grid de las imágenes estamos empleando un ancho y alto total mayor a las

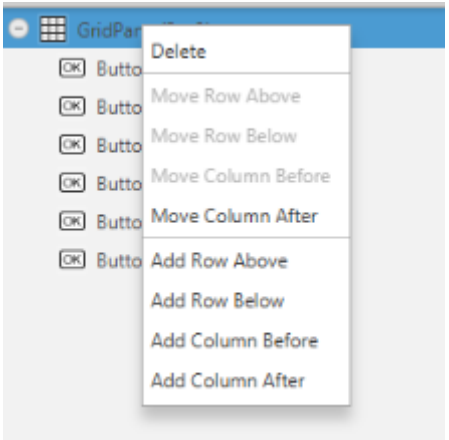
dimensiones de las filas y columnas. Es decir, el ancho total del grid es de 500px, mientras que cada columna tiene el valor **Pref Width** a 100px. Por tanto, el ancho restante se reparte automáticamente por defecto porque el valor de Hgrow es **SOMETIMES.** Sin embargo, las alternativas **ALWAYS** o **NEVER** permiten cambiar esta prioridad.



Como se puede observar en la imagen anterior, la columna 0 tiene el valor de Hgrow a ALWAYS, por lo que la columna 1 ocupa 100px y el resto se asigna a la columna 0 porque tiene mayor prioridad. Con NEVER ocurriría lo contrario y la prioridad se asigna al resto de columnas. La jerarquía sería **ALWAYS > SOMETIMES > NEVER**

El funcionamiento de Vgrow en las filas es exactamente igual.

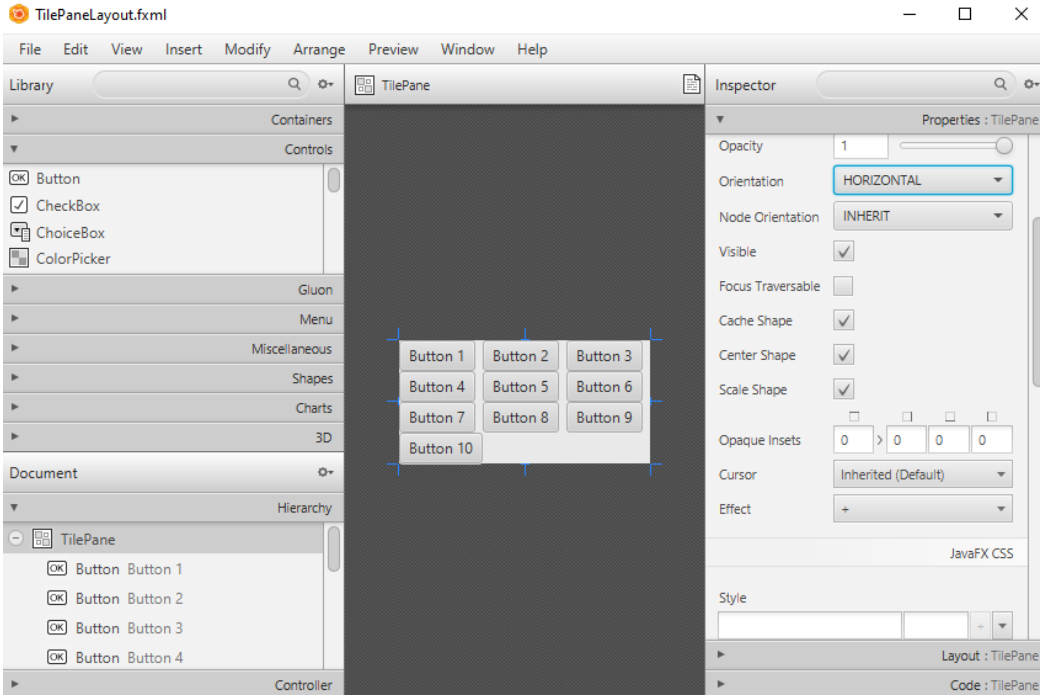
Por último, podemos añadir o quitar filas o columnas en un grid sobre el componente GridPane en el menú de la izquierda.



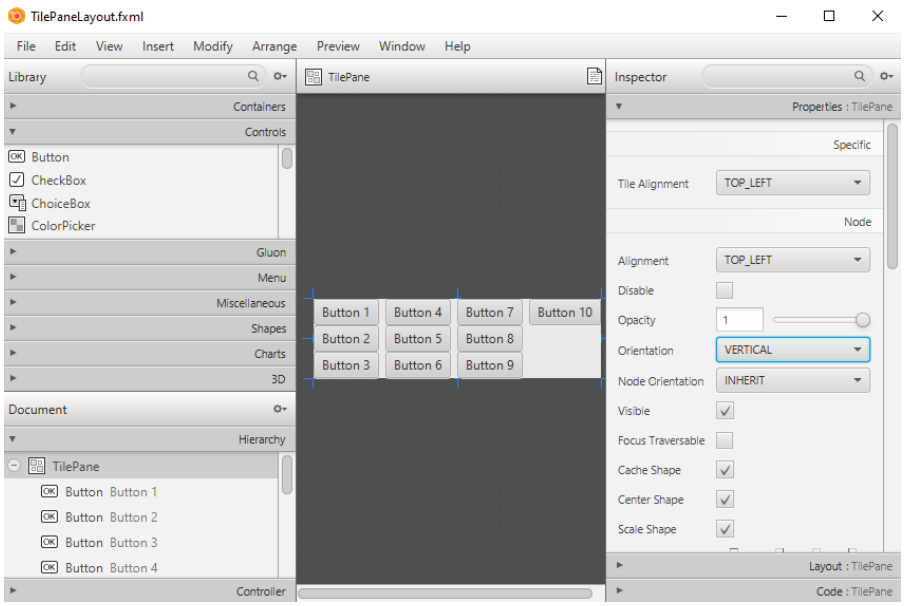
7. TilePane

TilePane al igual que FlowPane crea un layout que permite añadir nodos de forma vertical o horizontal. La diferencia radica en que TilePane puede especificar en filas y columnas como se van añadiendo los nodos.

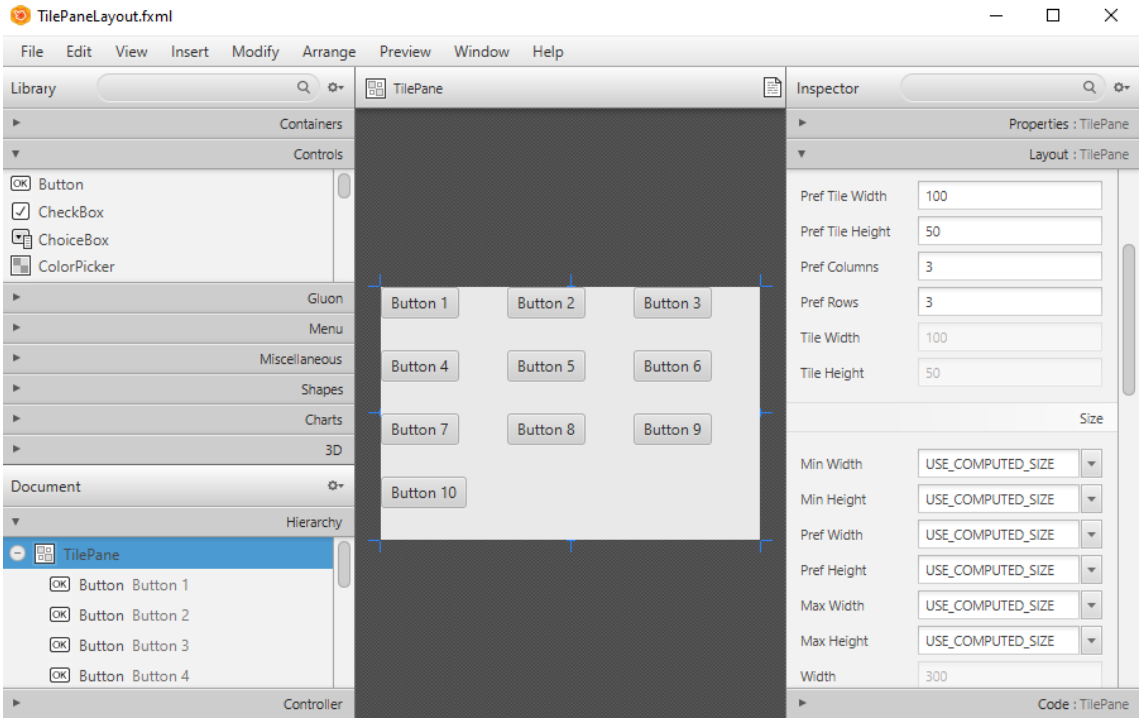
A continuación se añaden una serie de botones para ilustrar su funcionamiento en detalle.



Al igual que en **FlowPane**, se puede modificar la propiedad **Orientation**. En este caso las celdas se distribuyen de arriba a abajo en lugar de izquierda a derecha (también se puede cambiar el sentido con Node Orientation).
Si cambiamos la distribución a vertical, el resultado será el siguiente:



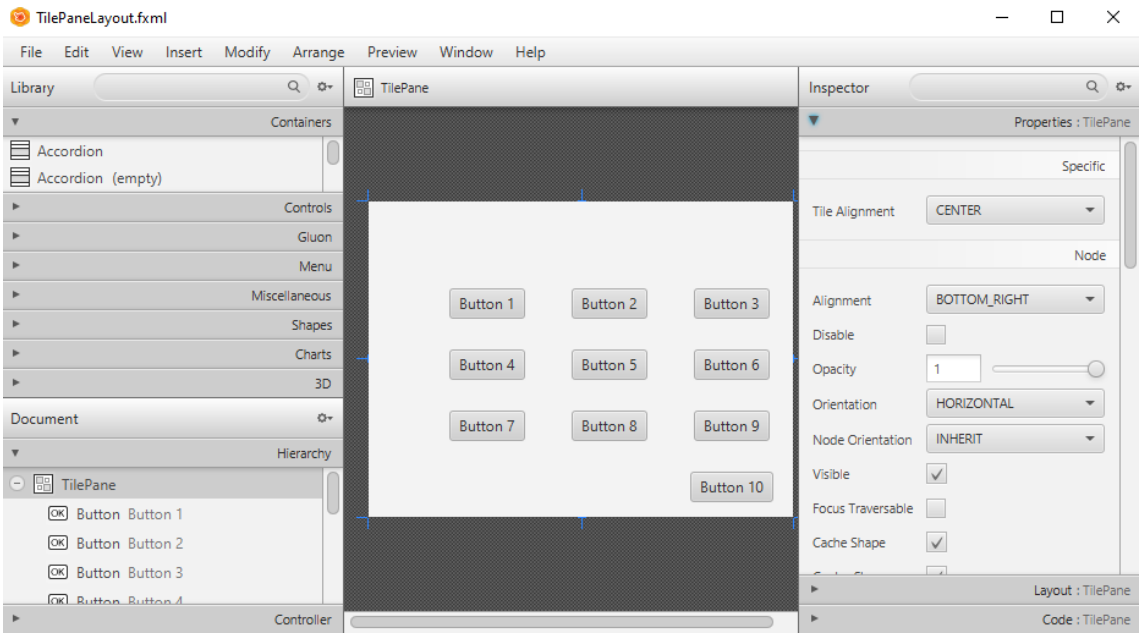
Dos propiedades claves para trabajar con este layout son **Pref Columns** y **Pref Rows.** En las imágenes de arriba, no se distribuían los botones en con 3 filas o 3 columnas por casualidad, sino porque el valor de ambas propiedades era 3. Por ello, JavaFX trata de distribuir el contenido en el número de filas y columnas especificadas. Prueba a modificar el número de filas y columnas y observa las diferencias.



No obstante, esto no es rígido y depende del tamaño de celda especificado en las propiedades **Pref Tile Width** y **Pref Tile Height** que en el ejemplo de la imagen de arriba se han modificado. Además, si modificamos **Pref Width** y **Pref Height**, estamos cambiando el tamaño del layout y esto implica que el número de celdas se modifique dinámicamente.

Si modificamos las propiedades especificadas en los párrafos anteriores, entonces se añade un espaciado entre celdas, ya que las dimensiones del layout aumentan. Por un lado, podemos añadir más espaciado con las propiedades **Padding**, **Hgap** o **Vgap** de la pestaña Layout.

Por último, también podemos modificar la alineación en la pestaña **Properties**, aunque solo será visible cuando las dimensiones permitan cierto espaciado entre celdas. **Alignment** se refiere a la alineación general del layout. Por ejemplo, en la imagen de debajo se ha seleccionado BOTTOM_RIGHT que tiende a desplazar los elementos abajo a la derecha. Mientras que **Tile Alignment** se refiere a la alineación del contenido dentro de cada fila o columna. En este caso se sitúan las filas y columnas en el centro del espacio que tenemos disponible. Prueba a modificar estas dos propiedades de alineación por otros valores y a cambiar las dimensiones del TilePane, de filas y columnas y añadir más espaciado para comprender las diferentes opciones.

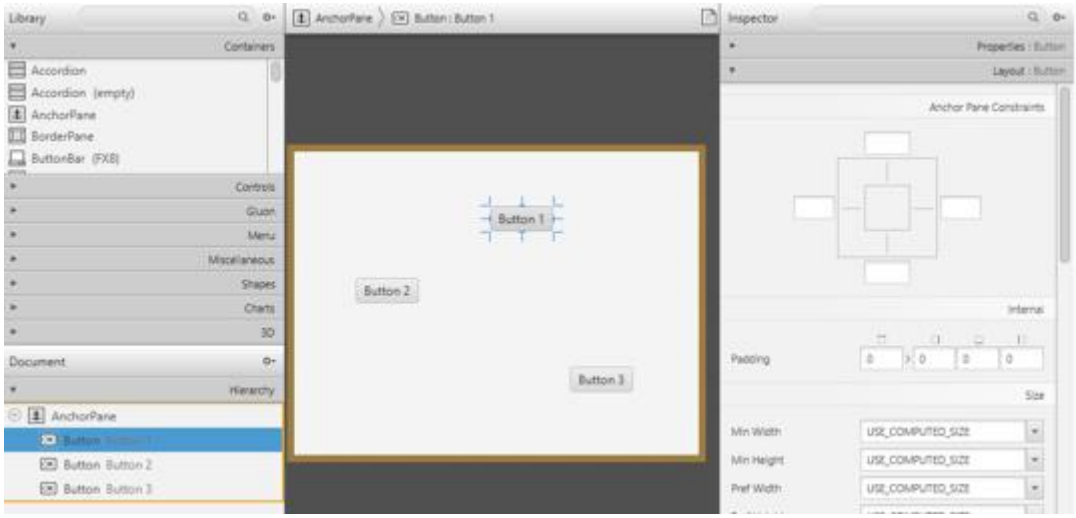


8. AnchorPane

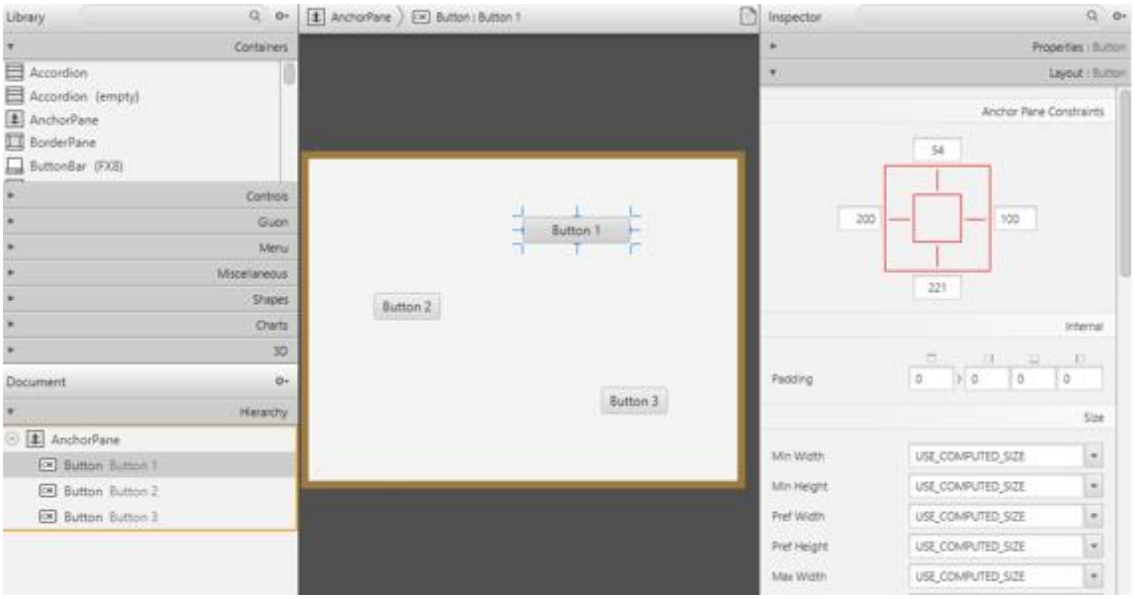
Es un layout es similar en cierto modo a **BorderPane**, pero con una serie de características que aportan mayor flexibilidad al posicionamiento de los elementos. Como su propio nombre indica, permite **anclar** elementos siempre en la misma posición.

En este caso, no tenemos secciones y los elementos se muestran inicialmente como si se tratara de un **StackPane.** Sin embargo, tenemos la libertad de arrastrar los componentes a la zona del **AnchorPane** que nos resulte adecuada para el diseño.

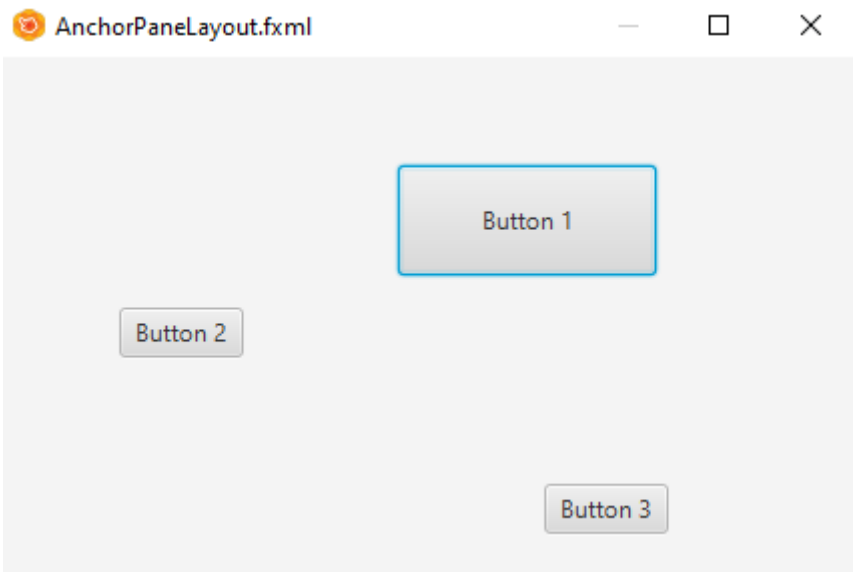
Por ejemplo, en la imagen de debajo hemos arrastrado tres botones a una posición específica. Si redimensionamos la pantalla los componentes con este layout siempre se quedan en la misma posición. Como se puede observar en la imagen de debajo, la propiedad **Anchor Pane Constraints** no tiene ningún valor y en este caso se quedarán los botones en las coordenadas iniciales a las que los hemos arrastrado.



Por otro lado, hay que tener cuidado al modificar las coordenadas que AnchorPane crea en la pestaña Layout de cada elemento incluido en el contenedor como se muestra debajo.



Si ahora arrancamos la aplicación, podemos observar que el layout fija el botón dejando siempre a cada lado la cantidad de píxeles indicadas en cada coordenada. Por ello, si agrandamos la ventana se ampliará o disminuirá el ancho o alto del botón para que se cumplan las restricciones de la propiedad **Anchor Pane Constraints** y que siempre haya 200 píxeles a la izquierda, 100 píxeles a la derecha, 54 píxeles por encima y 221 píxeles por debajo (lo que se especificó en el ejemplo de la imagen anterior).

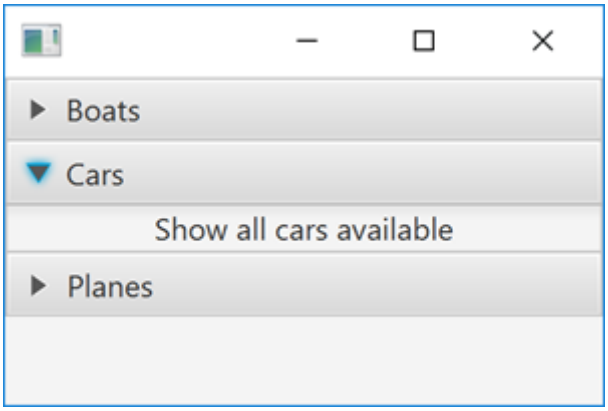


9. Otros layouts

Los patrones de diseño estudiados hasta ahora son los más básicos y lo mínimo necesario para desarrollar una aplicación con JavaFX. Sin embargo, existen otros layouts que permiten añadir mayor complejidad al diseño.

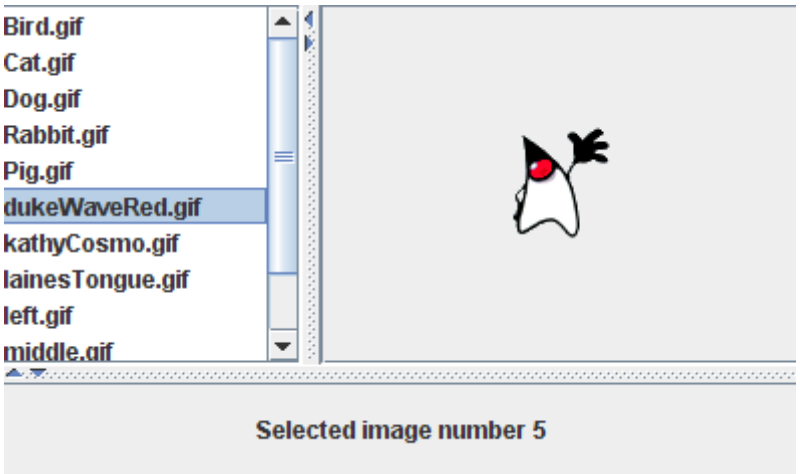
A continuación, se citan algunos:

- **Accordion:** Es un conjunto de secciones y solo una puede abrirse al mismo tiempo. De esta manera, se divide el diseño en varias pantallas independientes.



- **ScrollPane:** Es un patrón de diseño que añade un scroll a la página cuando se superan las dimensiones establecidas en el layout. Normalmente se utiliza como contenedor principal de la aplicación o con listados

- **SplitPane:** Permite dividir la pantalla en dos secciones independientes, ya sea de manera horizontal o vertical. También existe el SplitPane empty que permite añadir tantas divisiones como queramos.



- **TabPane:** Similar a Accordion, pero las secciones en este caso se separan por pestañas con botones en la parte superior de la página.