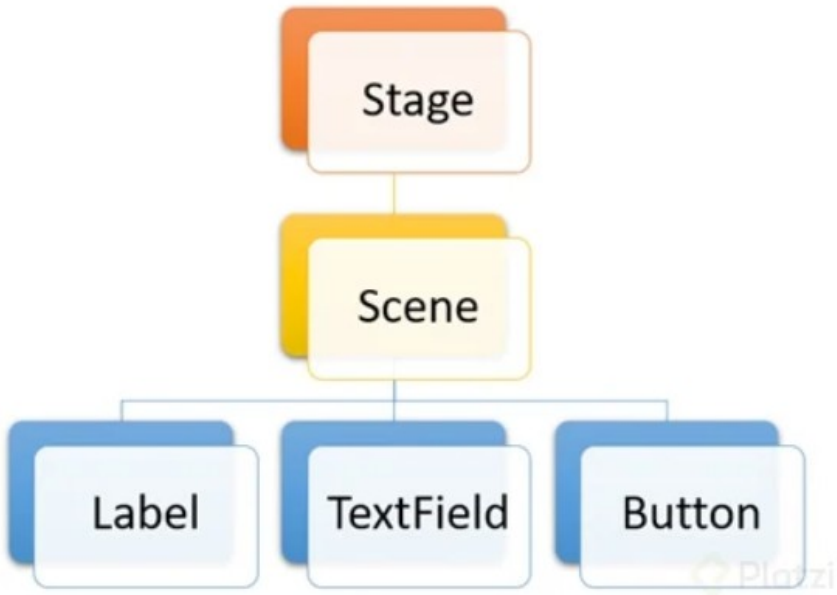
TEMA 2: JAVAFX

1 ¿QUÉ ES JAVAFX Y PARA QUÉ SIRVE?

JavaFX es una tecnología Java para el desarrollo de aplicaciones con interfaz gráfica interactivas multiplataforma para el escritorio y aplicaciones móviles. Está formada por un conjunto de clases y API junto con un editor gráfico Scene Builder para crear las interfaces visualmente.

Desde una perspectiva de Model View Controller (MVC):

- El archivo FXML, que contiene la descripción de la interfaz de usuario, es la vista.
- El controlador es una clase Java, que implementa opcionalmente la clase Initializable, que se declara como el controlador para el archivo FXML.
- El modelo consta de objetos de dominio, definidos en el lado de Java, que se pueden conectar a la vista a través del controlador.



Algunas de las características notables de JavaFX son:

- **Java APIs:** las APIs están escritas en código nativo Java compatibles con otros lenguajes soportados por la máquina virtual.
- **FXML and Scene Builder:** FXML es un lenguaje de marcado que describe las interfaces de usuario. Se pueden escribir directamente o usar la herramienta JavaFX Scene Builder para crearlos con una interfaz gráfica.
- **WebView**: permite embeber páginas HTML en las aplicaciones JavaFX. Ofrece soporte para JavaScript.

- **Built-in UI controls and CSS:** proporciona cantidad de controles para construir aplicaciones completas. El estilo de los controles puede ser modificado con CSS.
- **Canvas API:** para dibujar directamente en la pantalla.
- **Multitouch Support**: soporte para gestos táctiles múltiples en función de las posibilidades de la plataforma subyacente.
- **Hardware-accelerated graphics pipeline:** haciendo uso de la GPU se consiguen animaciones gráficas fluidas en las tarjetas gráficas soportadas, si la gráfica no está soportada de hace uso de la pila de software Java2D.
- **High-performance media engine:** soporta la reproducción de contenido multimedia con baja latencia basándose en GStreamer.
- **Self-contained application deployment model:** las aplicaciones contenidas tiene todos los recursos y una copia privada de los entornos de ejecución de Java y JavaFX. Son distribuidos como paquetes instalables y proporcionan la misma experiencia de instalación e inicio que las aplicaciones nativas del sistema operativo.
- **Aplicación MVC:** Las aplicaciones desarrolladas con JAVAFX, son un claro ejemplo del desarrollo ***bajo el patrón de diseño modelo vista controlador***, ya que estas dividen en el código de la interfaz gráfica completamente de la parte lógica de nuestra aplicación aquí juega un papel importante los ***archivos FXML***, cada vez que vean esta extensión simplemente piensen en interfaz gráfica.

2 ARCHIVOS FXML

Como su nombre lo indica, FXML es un formato basado en XML que permite a los desarrolladores definir interfaces de usuario de manera declarativa, en lugar de definir interfaces de manera programática, es decir, mediante el uso directo de las API de JavaFX.

FXML es un lenguaje de marcado basado en XML que permite a los desarrolladores para crear una interfaz de usuario (UI) en una aplicación JavaFX por separado de la aplicación de la lógica de la aplicación.

Para trabajar el aspecto de los controles podemos utilizar css de la misma forma que se hace en Html.

Proporciona funcionalidad multimedia ,disponible en todas las plataformas en las que se admite JavaFX.

JavaFX permite a los desarrolladores animar objetos gráficos en sus aplicaciones con mucha mayor facilidad que en swing, debido al escenario gráfico subyacente a la plataforma y las API de particulares que se crean específicamente para ese propósito.

Añadir contenido HTML a a las aplicaciones, pudiendo ademas programar la lógica de la aplicación con javascript

Si bien el archivo FXML se puede editar dentro del IDE, no se recomienda, ya que el IDE proporciona solo la comprobación de sintaxis básica y el autocompletado, pero no la guía visual. El mejor enfoque es abrir el archivo FXML con Scene Builder, donde todos los cambios se guardarán en el archivo.

(Instalación de Scence Builder). JavaFX Scene Builder es una herramienta de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario de aplicaciones JavaFX, sin necesidad de codificación. Los usuarios pueden arrastrar y soltar componentes de interfaz de usuario a

un área de trabajo, modificar sus propiedades, aplicar hojas de estilo y el código FXML del diseño que están creando se genera automáticamente en segundo plano. El resultado es un archivo FXML que se puede combinar con un proyecto Java vinculando la interfaz de usuario a la lógica de la aplicación.

La aplicación Scene Builder permite diseñar, mediante un interfaz gráfico, las estructuras de las ventanas de las aplicaciones que queramos desarrollar usando JavaFX. En este artículo podrás conocer los fundamentos básicos para empezar a usar esta herramienta de manera integrada con el entorno de desarrollo NetBeans.

- **El Controlador:** Las vistas de JavaFX carecen de cualquier comportamiento específico en la aplicación, y eso es bueno, porque mantiene las responsabilidades separadas.

  Los elementos de vista deberían ser responsables de definir cómo se ve la UI, pero no cómo se comporta la misma; esta es la responsabilidad de la parte del controlador.

  En términos de FXML, un **controlador** es un objeto que participa en DI y puede reaccionar a eventos activados por elementos de UI definidos en la vista asociada.

  A diferencia del punto de entrada principal de una aplicación JavaFX, donde es requerido extender una clase específica (javafx.application .Application), una clase de controlador puede definir su propia jerarquía sin necesidad de extender o implementar un tipo particular relacionado con JavaFX; esto te da rango libre para desarrollar tus propios tipos y jerarquías de controladores según sea necesario.

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

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

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

Este archivo FXML de ejemplo está asociado con una clase de controlador. La asociación entre el FXML y la clase del controlador, en este caso, se realiza especificando el nombre de la clase como el valor del atributo ***fx:controller*** en el elemento raíz del FXML:

fx:controller="com.example.FXMLDocumentController" .

Hace referencia a que nuestro archivo FXML va a tener un controlador

La clase del controlador permite que el código Java se ejecute en respuesta a las acciones del usuario en los elementos de la IU definidos en el archivo FXML

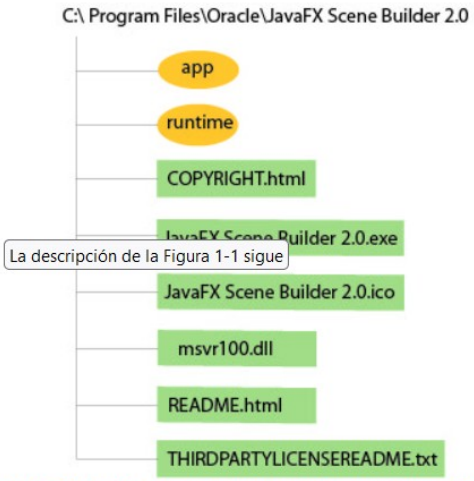
```
package com.mycompany.proyectojavafx;
import java.io.IOException;
import javafx.fxml.FXML;
public class PrimaryController {

    @FXML
    private void switchToSecondary() throws IOException {
        App.setRoot("secondary");
    }
}
```

3 CREAR LA PRIMERA APLICACIÓN JAVAFX

Antes de poder ejecutar una aplicación JavaFX de muestra, es necesario tener las bibliotecas de tiempo de ejecución de JavaFX en su máquina. Antes de continuar con estos pasos, instale la última versión del JDK o la última versión del JRE.

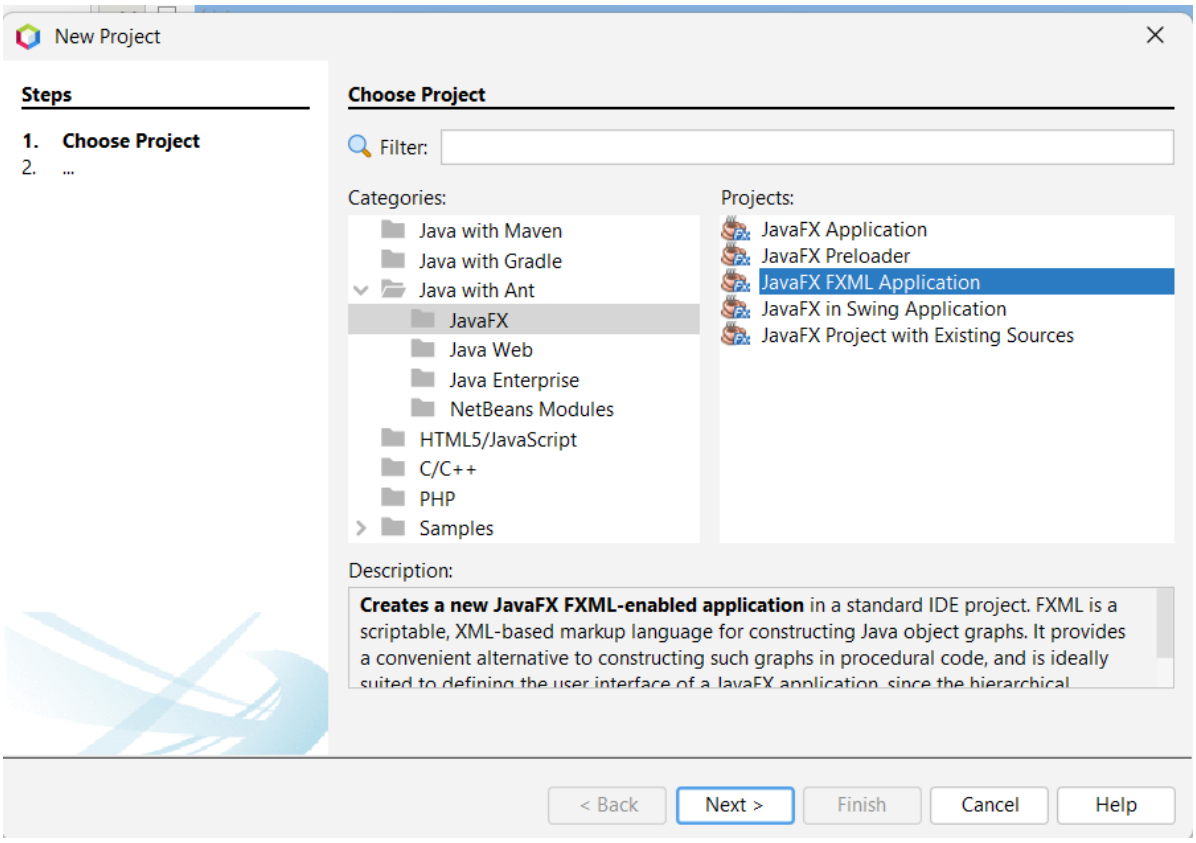
Figura 1-1 Contenido de una instalación de JavaFX Scene Builder 2.0 en una plataforma Windows



Descripción de "Figura 1-1 Contenido de una instalación de JavaFX Scene Builder 2.0 en una plataforma Windows"

Puede utilizar varios IDEs de desarrollo Java para desarrollar aplicaciones JavaFX. Los siguientes pasos explican cómo ver y ejecutar el código fuente en el IDE NetBeans.

- Desde el NetBeans abrir un nuevo proyecto



- Para crear una aplicación JavaFX solamente debemos extender la clase **Application** y sobrescribir el método **start**(Stage primaryStage), este es el punto de inicio de nuestra aplicación, el **Stage** llamado primaryStage es creado automáticamente y representa nuestra ventana principal o primaria, luego podremos crear más ventanas si lo deseamos, un **Stage** es similar a lo que sería un JFrame en Swing.

```
package com.mycompany.proyectojavafx;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;

    @Override
    public void start(Stage stage) throws IOException {
        //Estoy cargando mi archivo fxml principal como el nodo raiz (parent)

        scene = new Scene(loadFXML("primary"), 640, 480);
```

```
        //El objeto Stage, es la ventana en la cual agregaremos todos los objetos de
nuestra
        //interfaz gráfica está actuará como contenedor principal
        //Establece la escena principal
        stage.setScene(scene);
        //Desplegamos la ventana
        stage.show();
    }

@Override
  public void stop() throws Exception { }

}
    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml
+ ".fxml"));
        return fxmlLoader.load();
    }

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

}
```

Existen tres métodos que podemos sobrescribir para administrar el ciclo de ejecución de una aplicación JavaFX, el primer método en ser llamado es ***public void init()***, este se ejecuta antes de llamar al método ***public void start(Stage primaryStage)*** y nos sirve para tareas de inicialización, al detener la aplicaciíon se llama el ***método public void stop()***, nos sirve para tareas de finalización y liberación de recursos.

Una aplicación JavaFX no requiere del método main() para iniciar la aplicación, sin embargo este método puede ser requerido por algunos IDEs para identificar el punto de partida de la aplicaciíon.

Si deseamos iniciar la aplicación desde el método main() es necesario llamar al método ***Application.launch()*** que es el encargado de iniciar la apliación JavaFX.

- **launch()-→** Método que recibe los parámetros que podemos pasarle a nuestra apliación. Método que dispara la ejecución de la aplicación
- **Mostrar la Ventana:**

  La clase **Stage** nos permite manejar las ventanas creada por la API JavaFX, en primera instancia contamos con el **Stage** creado automáticamente por JavaFX, para mostrarlo usaremos el el método **primaryStage.show()**,

  ***stage.setScene(scene)* –>** Establece la escena principal
  stage.show() –> Mostrar

En JavaFX un **Stage** representa nuestra ventana, esta ventana puede contener solo un **Scene**, este es un contenedor para los elementos que conforman la GUI, puede contener uno o varios elementos organizados de manera jerárquica en forma de Arbol, a esto se le llama Scene Graph.

Los elementos que contiene un **Scene** deben extender la clase base **Node**, estos son organizados de manera jerárquica en el scene graph, los elementos incluidos en el scene graph son llamados **nodes**, al elemento ubicado en el tope de la jerarquía es el nodo raiz, este puede contener otros nodos hijos y estos a su vez otros nodos. Se crea una estructura de árbol jerárquica.

La clase **javafx.scene.layout.Pane** proporciona las bases para las clases que normalmente son utilizadas como contenedores, por ejemplo: **VBox, GridPane, StackPane**, otros, estos contenedores nos facilitan la organización de los nodos que agregamos a la GUI, normalmente elegimos a uno de ellos para utilizarlo como root node.

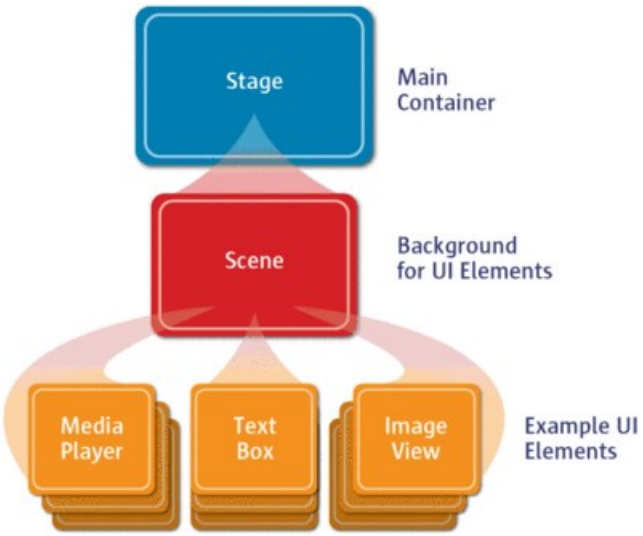
Para establecer el root del scene podemos usar el constructor o el método **setRoot(Parent)**, no necesariamente debe ser un contenedor, podemos usar cualquier clase que herede de la clase **javafx.scene.Parent** por ejemplo un Button o un Group.

4 ESTRUCTURA DE UNA APLICACIÓN JAVAFX

Una aplicación desarrollada con JavaFX está compuesta por tres componentes esenciales los cuales son:

- **Stage** - Escenario.
- **Scene** - Escena.
- **Nodes** - Nodos.

A continuación ilustramos en un diagrama la estructura de estos tres componentes que juntos son la esencia de una aplicación desarrollada con JavaFX.



4.1 Clase Stage.:

Esta clase nos provee de varios componentes importantes un **Stage** o **Escenario** en sí es una ventana en la cual agregaremos todos los objetos de nuestra interfaz gráfica está actuará como contenedor principal, actúa de manera similar a lo que ya conocemos en Java Swing con la clase Jframe.

El escenario o **Stage** principal es creada por la plataforma en si. El objeto tipo **Stage** es creado y se pasa como argumento al método start () de la clase Application, para poder visualizar dicha ventana debemos hacer uso del método show(), es importante resaltar que una aplicación JavaFX puede tener múltiples objetos Stage.

Para poder agregar objetos a este contenedor se necesitarán otro tipo de componentes como los **Scene** o los **Scene Graph**

4.2 Clase Scene

Cuando nos referimos a una escena o **Scene**, estamos haciendo referencia a los o contenidos físicos (nodos) de una aplicación JAVAFX, la clase **Scene** perteneciente al paquete Javafx.scene proporciona todos los métodos para manejar un objeto de escena. la creación de una escena al momento del desarrollo de cualquier aplicación con esta tecnología es necesaria para visualizar los contenidos en el escenario(Stage), al crear un objeto de tipo **Scene**, debemos pasar como argumento el nodo raíz el cual contendrá todos los nodos que se visualizarán en dicha interfaz e igualmente tenemos la posibilidad de asignar unas dimensiones(ancho y alto) a la escena.

4.3 Nodos Gráficos de Escena

Un nodo es un componente gráfico básico de una escena. Donde una escena es la representación visual de los componentes (el contenido de la ventana). Dentro de un escenario hay un árbol de elementos, estos elementos son llamados nodos, los nodos pueden ser leaf(hoja), que no se pueden agregar otros nodos a ellos o pueden ser una branch (rama), que pueden contener mas nodos.

Cada elemento dentro de una escena es un nodo. Los nodos hoja tienen cero sub-elementos y los nodos rama tienen cero o mas sub-elementos.

Dentro de un árbol de escenario solo habrá un nodo que no tendrá padre/rama, al que se le llama root o raíz.

Una escena es quien contiene el contenido visual mientras que un escenario es quien contiene la escena. En otras palabras, un stage seria el escenario y una scene seria la escena.

Dentro de Parent Existen dos clases que nos importan mucho, la clase **Control** y la clase **Region**. Ya que de **Control** heredan los componentes visuales mas usados: botones, etiquetas, menú, etc. De **Region** heredan los componentes de distribución o layouts, los cuales nos permiten "acomodar" otros componentes.

5 CLASE CONTROLADORA

Todos los componentes que queramos controlar desde nuestra clase controladora debemos etiquetarlos con la etiqueta fxml y el nombre debe ser el mismo nombre con el que ha sido declarado en el archivo fxml.

6 LAYOUTS

Un Layout en JAVAFX, lo podríamos definir como la manera en que serán distribuidos los distintos nodos a través de nuestra ventana o interfaz gráfica.

6.1 LAYOUTS JAVA SWING vs LAYOUTS JAVAFX

Cuando trabajamos con JAVA SWING, se creaba un JPanel al cual debiamos asignarle un Layout de cualquier tipo(Flowlayout, Gridbaglayout, BorderLayout, etc...) de lo contrario se entendía que teníamos un Layout nulo(null) y posicionamos los elementos a través de coordenadas.

Cuando trabajamos con JAVAFX, trabajamos de manera similar por lo menos en los tipo de Layouts existentes, pero no manejamos clases separadas como en Swing, la cual una clase pertenecía al contenedor y la otra al Layout, con JAVAFX se maneja una clase la cual sirve como contenedor y Layout y esto se debe a que dicha clase hereda de la clase Pane la cual es la clase padre de todos los contenedores de JAVAFX, les recomiendo revisen la documentación oficial para reforzar conocimientos,

6.2 TIPOS DE LAYOUTS

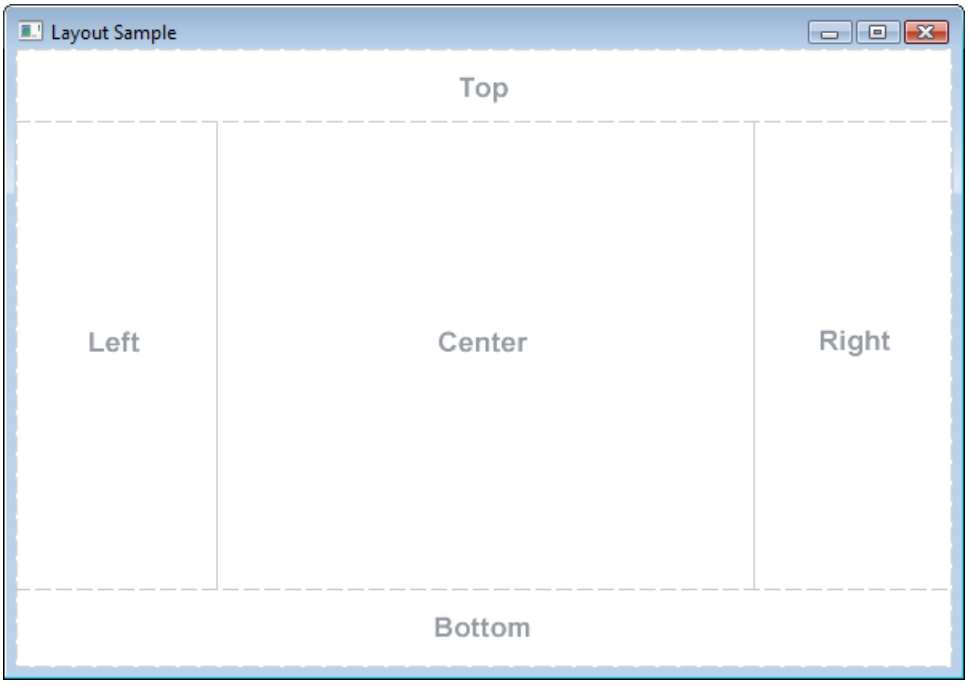
JAVAFX nos provee de varios Contenedores de tipo Layout, los cuales iremos definiendo uno a uno en cada entrada, por el momento enumeramos los tipos de Layouts existentes, el dia de hoy iniciaremos con el BorderPane:

- BorderPane
- HBox
- VBox
- StackPane
- TextFlow
- AnchorPane
- TitlePane
- GridPane
- FlowPane

6.3 BORDERPANE LAYOUT

Este tipo de Layout permite agregar o dividir nuestro contenedor en 5 regiones las cuales son, arriba(top), abajo(bottom), izquierda(left), derecha(right), centro(center).

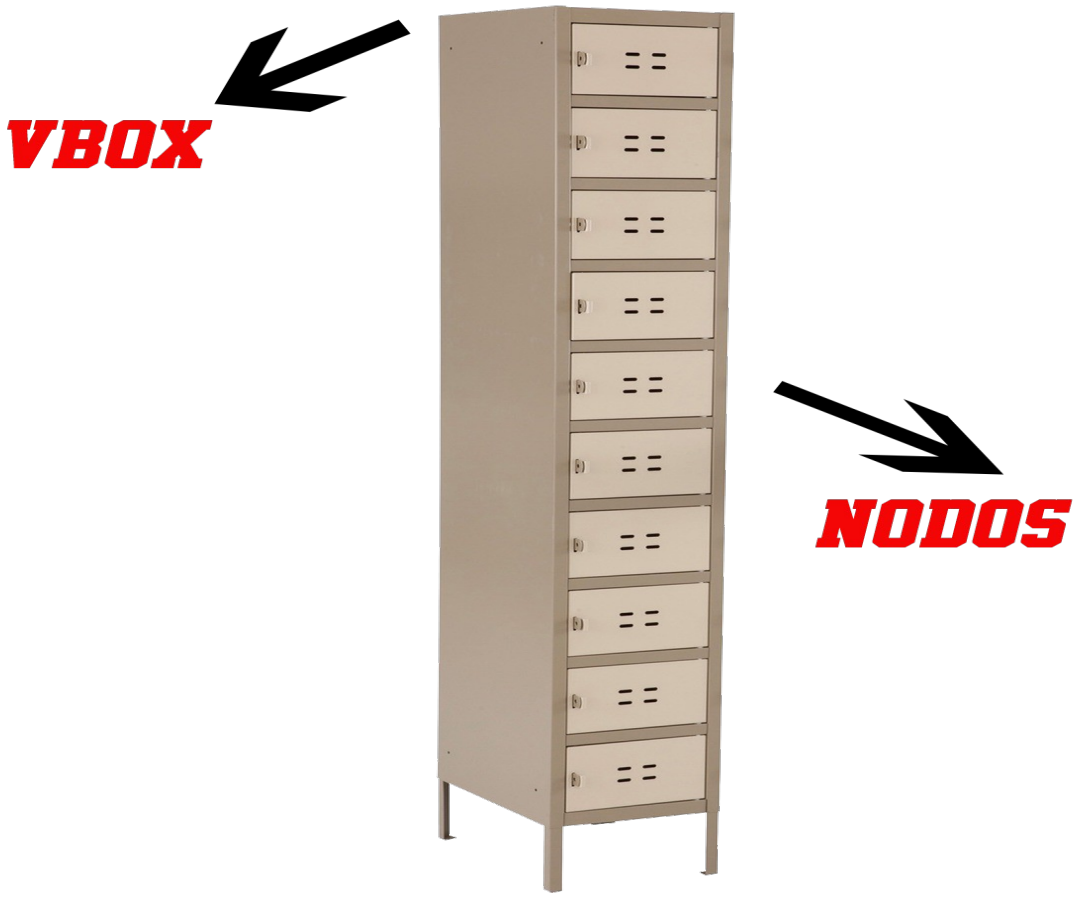
Por lo tanto al momento de agregar Nodos hijos(botones, etiquetas, tablas) a este tipo de layout solo se podrá hacer en estas posiciones anteriormente mencionadas.



6.4 VBOX

Cuando utilizamos un panel de diseño de tipo VBox todos los elementos hijos se mostrarán en una sola columna vertical.

Siempre que voy a explicar este tipo de Layout lo comparo con una columna o un locker donde guardamos objetos de todo tipo por ejemplo; zapatos, libros etc... podríamos comparar este tipo de layout como una matriz de 1 columna por n filas, el número de filas en este caso sería equivalente al número de nodos o componentes que vamos a utilizar, cada nodo que agregue-mos será ordenado uno debajo del otro.



Es importante resaltar que esta clase pertenece al paquete javafx.scene.layout, igualmente esta clase heredada de la clase Panel la razón es porque todos los paneles de tipo Layouts heredan de esta clase, esto lo estuvimos tratando en la entrada anterior

- **alignment:** esta propiedad representa la alineación de los nodos dentro de los límites de VBox. esta propiedad se establece mediante el método setAlignment (), podemos posicionar los elementos al centro, a la izquierda, derecha etc.
- **fillHeight:** esta propiedad es de tipo booleano y al establecer esto como verdadero;Los nodos redimensionables en el VBox se redimensionan a la altura del Vbox. Puede establecer el valor de esta propiedad utilizando el método setter setFillHeight().
- **spacing:** esta propiedad la utilizamos para agregar un espacio entre nodos y de esta manera crear una separación entre los mismos, para acceder a esta propiedad lo hacemos a traves del metodo setSpacing().
- **setVgrow:** al utilizar este metodo obligamos a un nodo hijo a redimensionarse una vez el VBox cambie sus dimensiones, es importante resaltar que el crecimiento del nodo hijo será vertical.
- **setMargin:** con este método podemos aplicar márgenes al VBox, con este método podemos aplicar márgenes a un nodo hijo específicamente.

Bibliografia: curso practico JavaFX (internet)
Bibliografia:Tutorial JavaFX (internet)