



IES Sant Vicent Ferrer
Algemesí



**GENERALITAT
VALENCIANA**
CONSELLERIA D'EDUCACIÓ,
INVESTIGACIÓ, CULTURA I ESPORT



Fondo Social Europeo

Programación

UD 12: Interfaz Gráfica de Usuario

- JavaFX -

Ciclo Formativo de Grado Superior
Desarrollo de Aplicaciones Web

Jose Chamorro Molina
Revisado por: J. Ramón Simó

JavaFX

¿Qué es JavaFX?

JavaFX es un conjunto de paquetes para gráficos y tecnologías de Oracle Corporation (inicialmente Sun Microsystems), que permite a los desarrolladores diseñar, crear, probar, depurar e implementar aplicaciones cliente que operan de manera consistente en diversas plataformas.

Con JavaFX diseñaremos de una forma fácil e intuitiva aplicaciones de escritorio profesionales.



Nota

JavaFX es una tecnología muy extensa para profundizar en este curso. El objetivo será, por tanto, estudiar los conceptos básicos para crear aplicaciones con interfaces sencillas en JavaFX.

Instalación y configuración

Instalar JavaFX

¡¡Atención!!

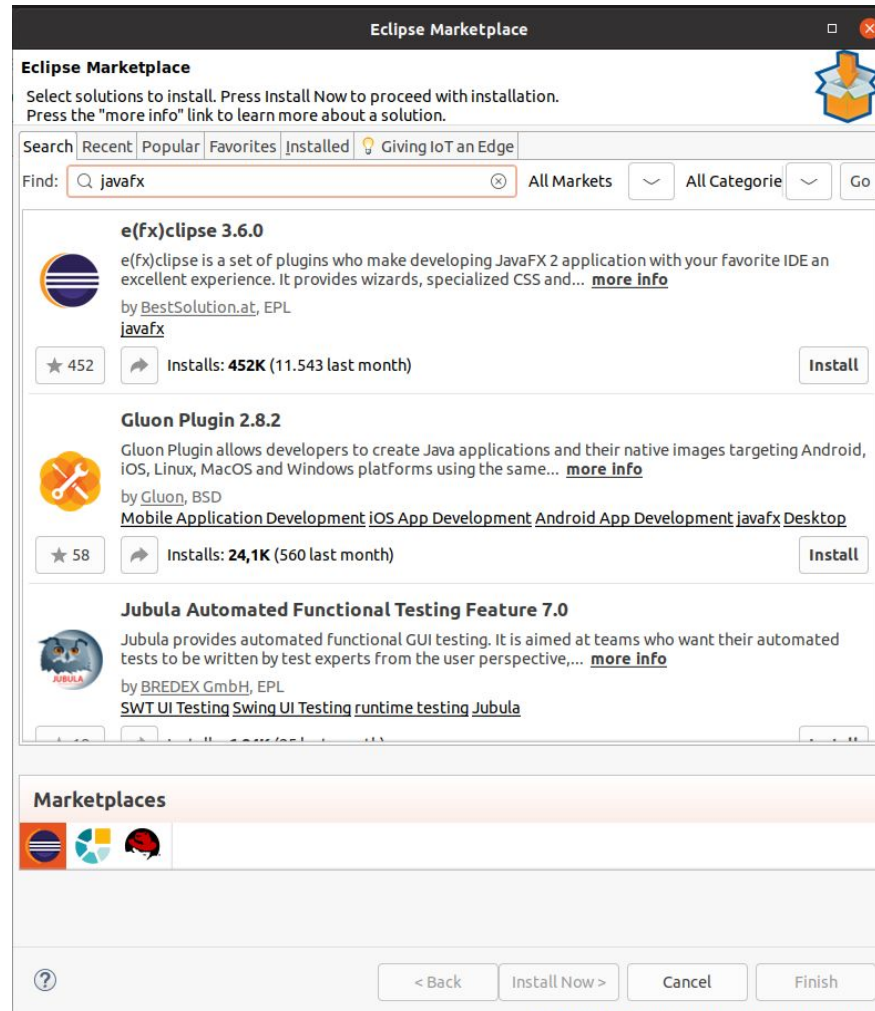
Se ha comprobado que ciertas funcionalidades del plugin JavaFX no funcionan con la versión más reciente de Eclipse (2023-04 (4.27)).

Por tanto, se recomienda tener instalada la versión **2022-09 (4.25)** de Eclipse para poder realizar las actividades de esta unidad.

Instalar JavaFX

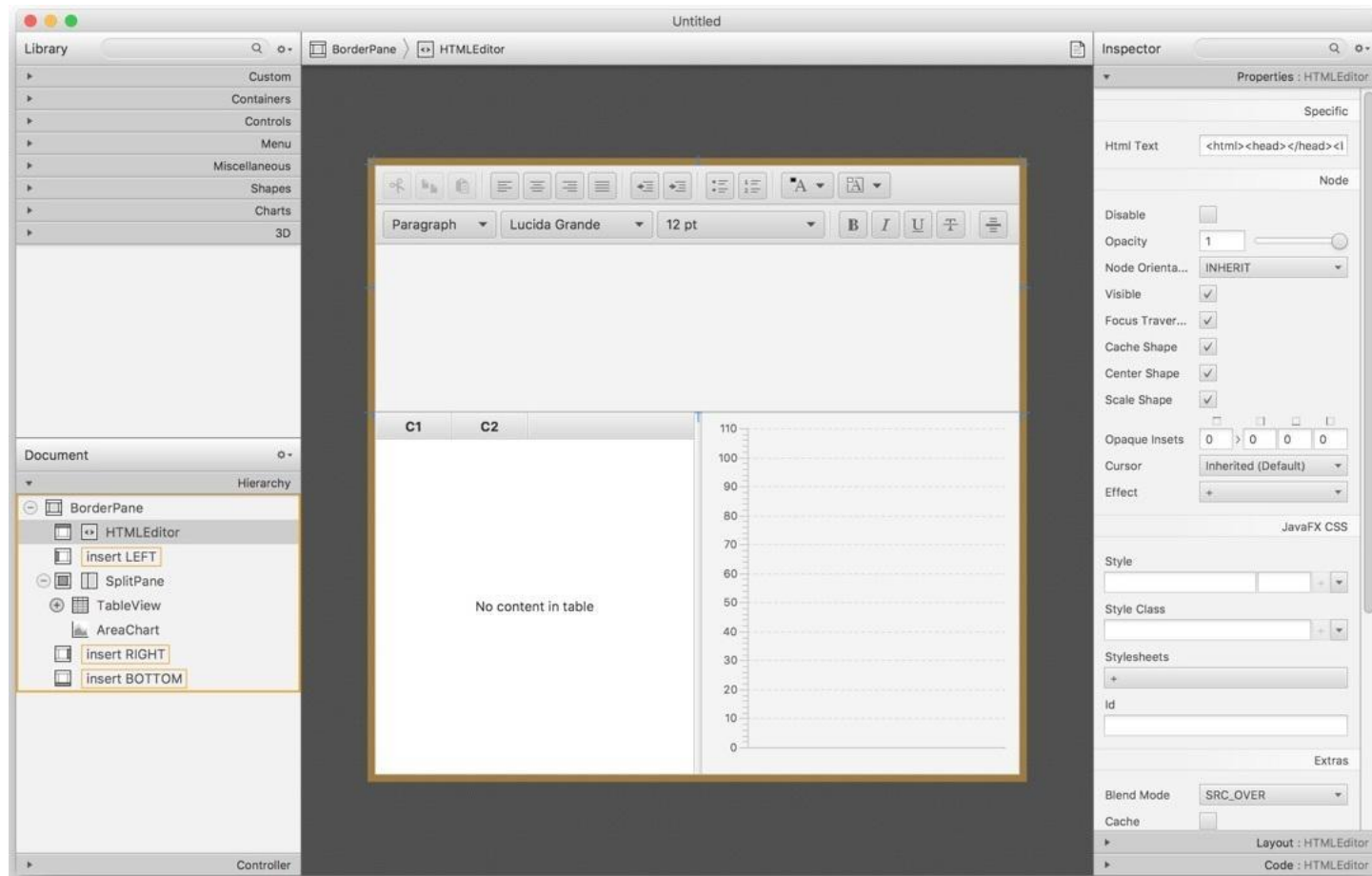
help > Eclipse Marketplace > find “javafx”

e(fx)clipse 3.8.0 > Install



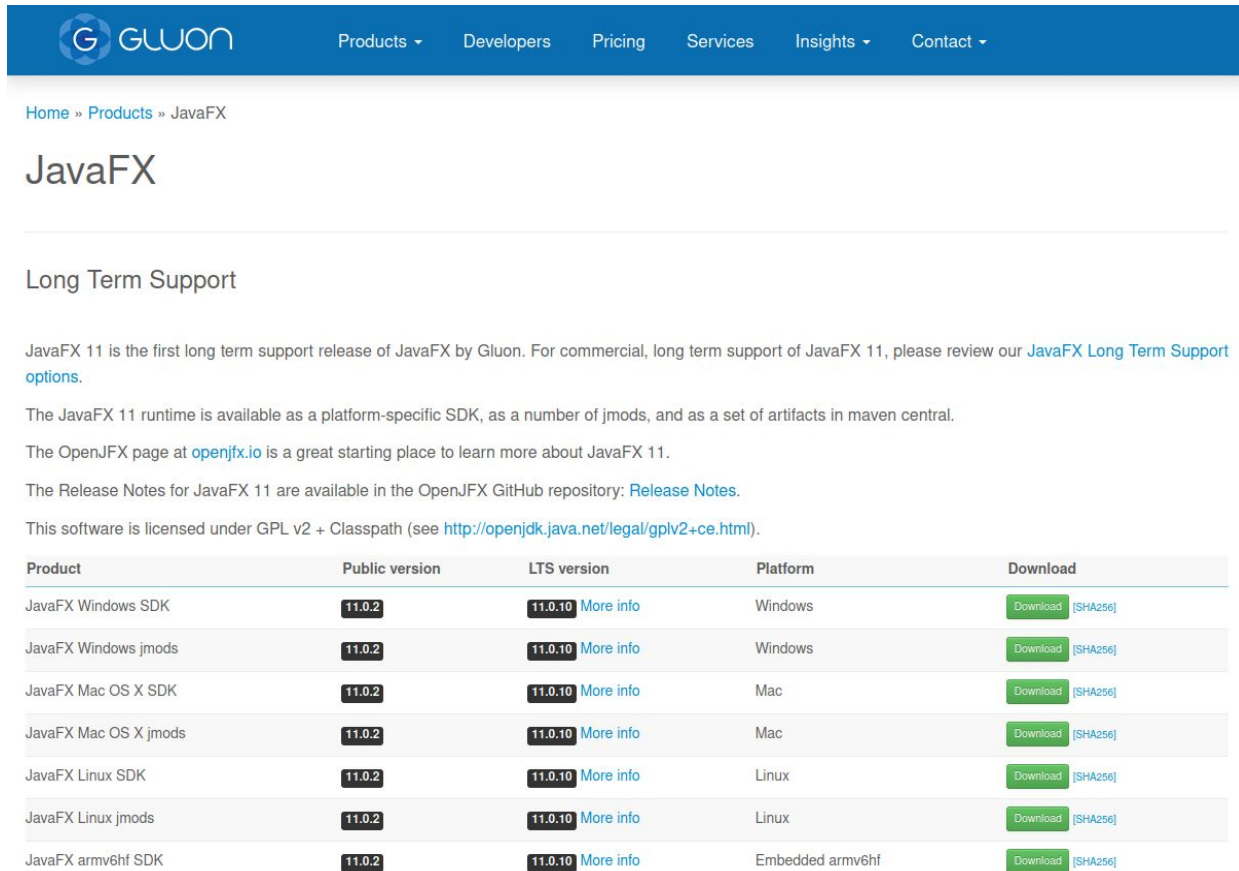
Instalar SceneBuilder

<https://gluonhq.com/products/scene-builder/>



Descargar SDK JavaFX

<https://gluonhq.com/products/javafx/>



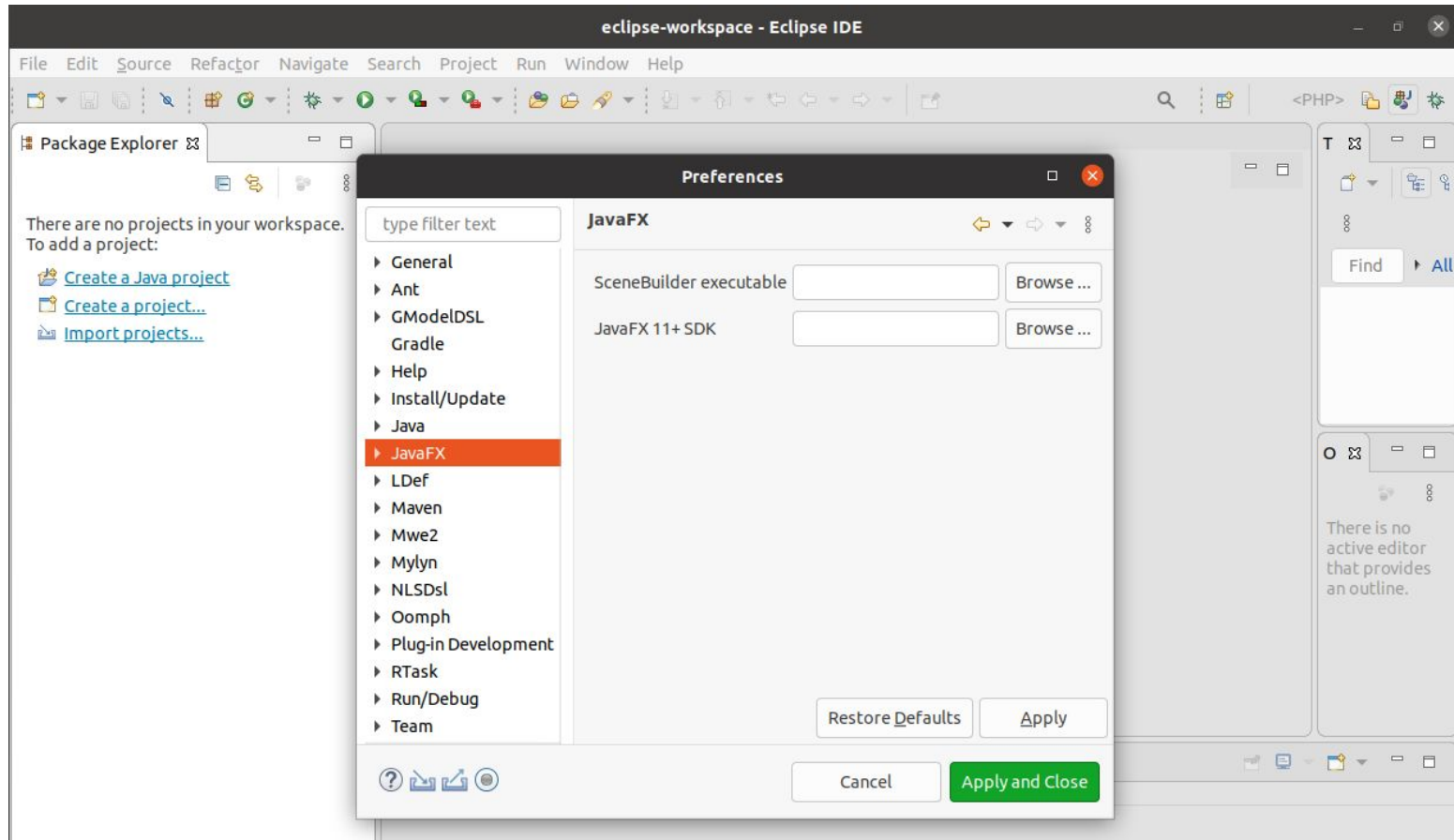
The screenshot shows the Gluon JavaFX download page. The header is blue with the Gluon logo and navigation links: Products, Developers, Pricing, Services, Insights, and Contact. The breadcrumb trail is Home » Products » JavaFX. The main heading is 'JavaFX'. Below it, the section 'Long Term Support' is highlighted. The text explains that JavaFX 11 is the first long term support release and provides links for more information, including 'JavaFX Long Term Support options', 'openjfx.io', and 'Release Notes'. It also mentions the license (GPL v2 + Classpath). At the bottom, there is a table with columns: Product, Public version, LTS version, Platform, and Download. The table lists various SDKs and jmods for Windows, Mac OS X, Linux, and Embedded armv6hf, each with a download button and a SHA256 link.

Product	Public version	LTS version	Platform	Download
JavaFX Windows SDK	11.0.2	11.0.10 More info	Windows	Download [SHA256]
JavaFX Windows jmods	11.0.2	11.0.10 More info	Windows	Download [SHA256]
JavaFX Mac OS X SDK	11.0.2	11.0.10 More info	Mac	Download [SHA256]
JavaFX Mac OS X jmods	11.0.2	11.0.10 More info	Mac	Download [SHA256]
JavaFX Linux SDK	11.0.2	11.0.10 More info	Linux	Download [SHA256]
JavaFX Linux jmods	11.0.2	11.0.10 More info	Linux	Download [SHA256]
JavaFX armv6hf SDK	11.0.2	11.0.10 More info	Embedded armv6hf	Download [SHA256]

- Descomprimir en una carpeta de fácil acceso

Configurar Eclipse

- Especificar la ruta al ejecutable de SceneBuilder
- Especificar la ruta al “/lib” de la carpeta descomprimida del SDK de JavaFX



Nuestro primer proyecto

Crear Proyecto

Crear proyecto tipo JavaFX:

New -> Other... ->
JavaFX -> JavaFX Project

The screenshot shows the 'New Java Project' dialog box in the Eclipse IDE. The dialog is titled 'New Java Project' and has a subtitle 'Create a Java project in the workspace or in an external location.' The main content area is divided into several sections: 'Project name' with a text field containing 'HolaMundoFX'; 'Use default location' checked, with a 'Location' field showing 'C:\Users\direccio\workspace\HolaMundoFX' and a 'Browse...' button; 'JRE' section with three radio buttons: 'Use an execution environment JRE:' (selected) with a dropdown showing 'JavaSE-1.8', 'Use a project specific JRE:' with a dropdown showing 'jre', and 'Use default JRE 'jre' and workspace compiler preferences' with a 'Configure JREs...' link; 'Project layout' section with two radio buttons: 'Use project folder as root for sources and class files' (selected) and 'Create separate folders for sources and class files' with a 'Configure default...' link; 'Working sets' section with a checkbox 'Add project to working sets' and a 'New...' button, and a 'Working sets:' dropdown with a 'Select...' button; 'Module' section with a checkbox 'Create module-info.java file'; and a footer section with a help icon, a '< Back' button, a 'Next >' button, an 'Finish' button, and a 'Cancel' button. A note at the bottom states: 'The default compiler compliance level for the current workspace is 16. The new project will use a project specific compiler compliance level of 1.8.'

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: HolaMundoFX

☒ Use default location

Location: C:\Users\direccio\workspace\HolaMundoFX [Browse...](#)

JRE

☒ Use an execution environment JRE: JavaSE-1.8

☐ Use a project specific JRE: jre

☐ Use default JRE 'jre' and workspace compiler preferences [Configure JREs...](#)

Project layout

☒ Use project folder as root for sources and class files

☐ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets [New...](#)

Working sets: [Select...](#)

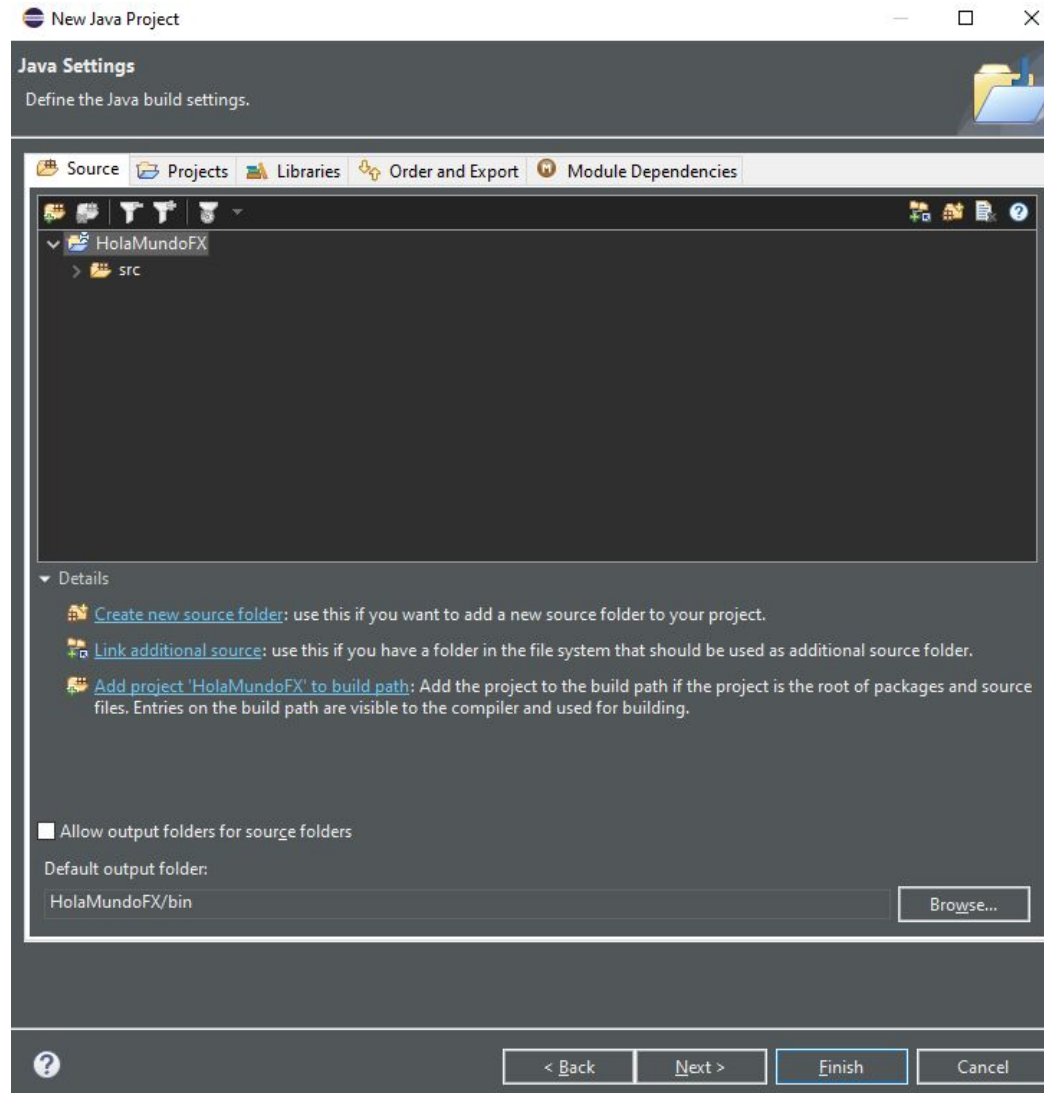
Module

☐ Create module-info.java file

i The default compiler compliance level for the current workspace is 16. The new project will use a project specific compiler compliance level of 1.8.

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

Crear Proyecto

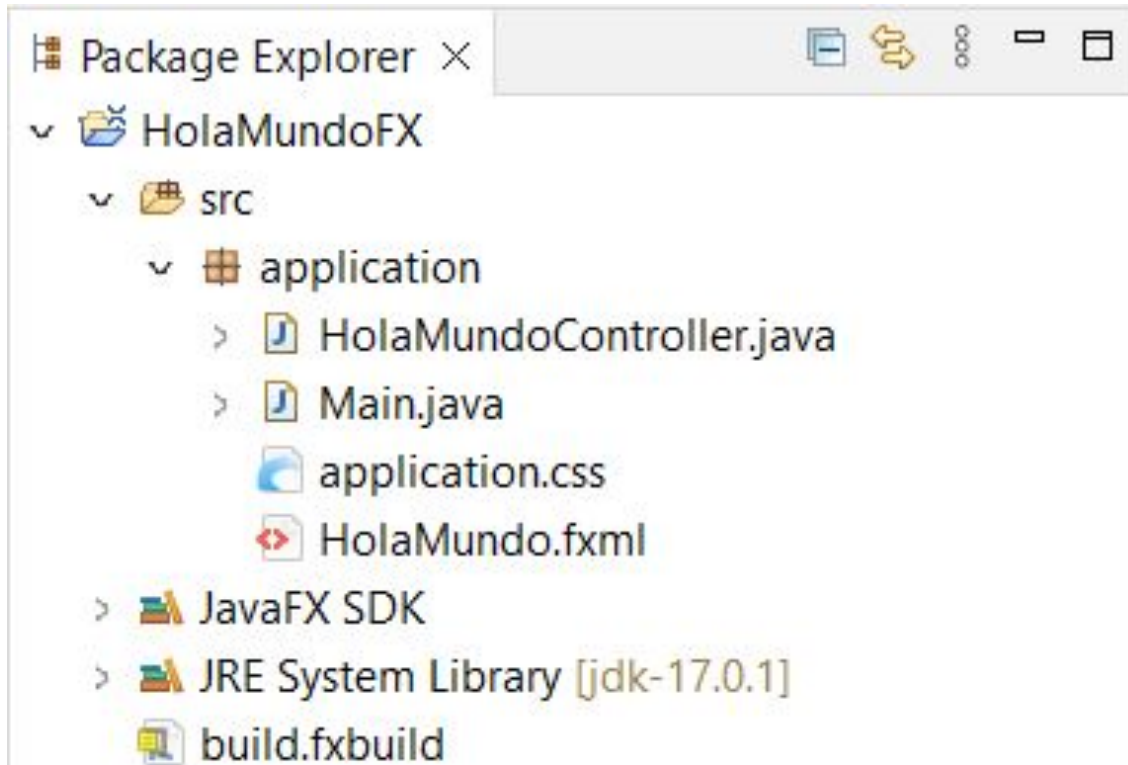


Crear Proyecto



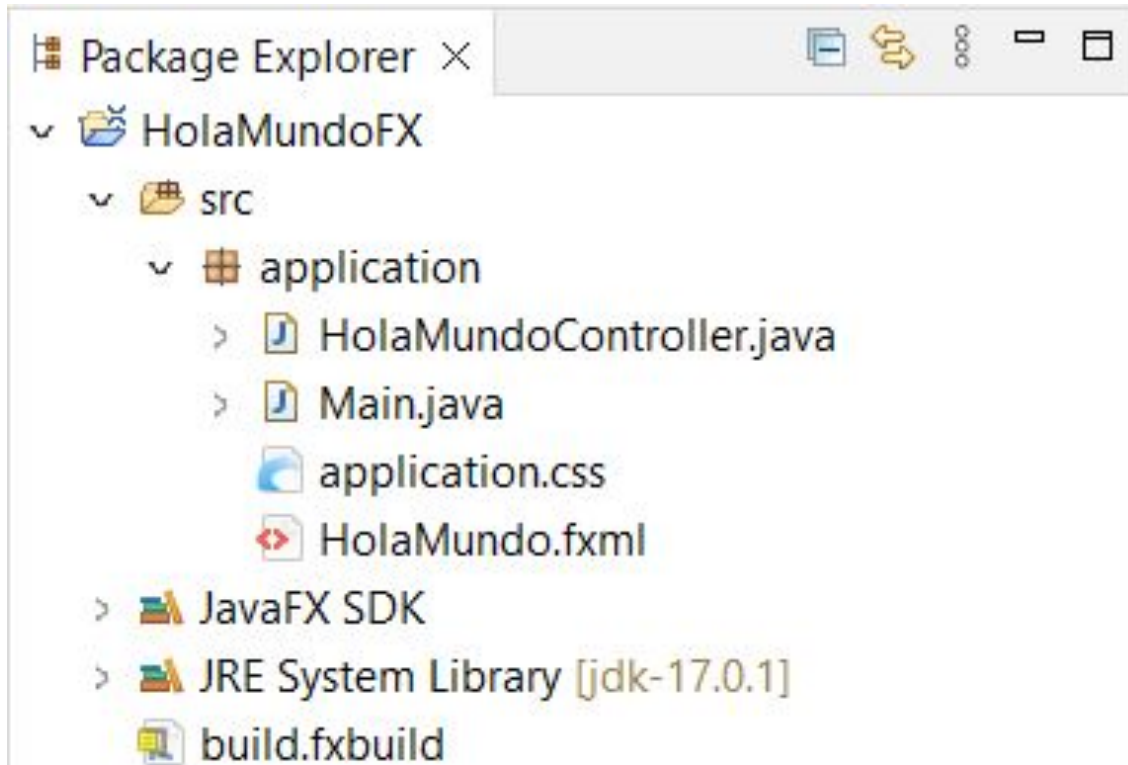
Estructura del Proyecto

Crear los siguientes paquetes por defecto:



Estructura del Proyecto

Crear los siguientes paquetes por defecto:



Main.java

Clase Main por defecto crea el método **start** en vez de **main** como punto de ejecución de la App:

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            AnchorPane root = (AnchorPane)FXMLLoader.Load(getClass().getResource("HolaMundo.fxml"));
            Scene scene = new Scene(root,400,400);
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

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

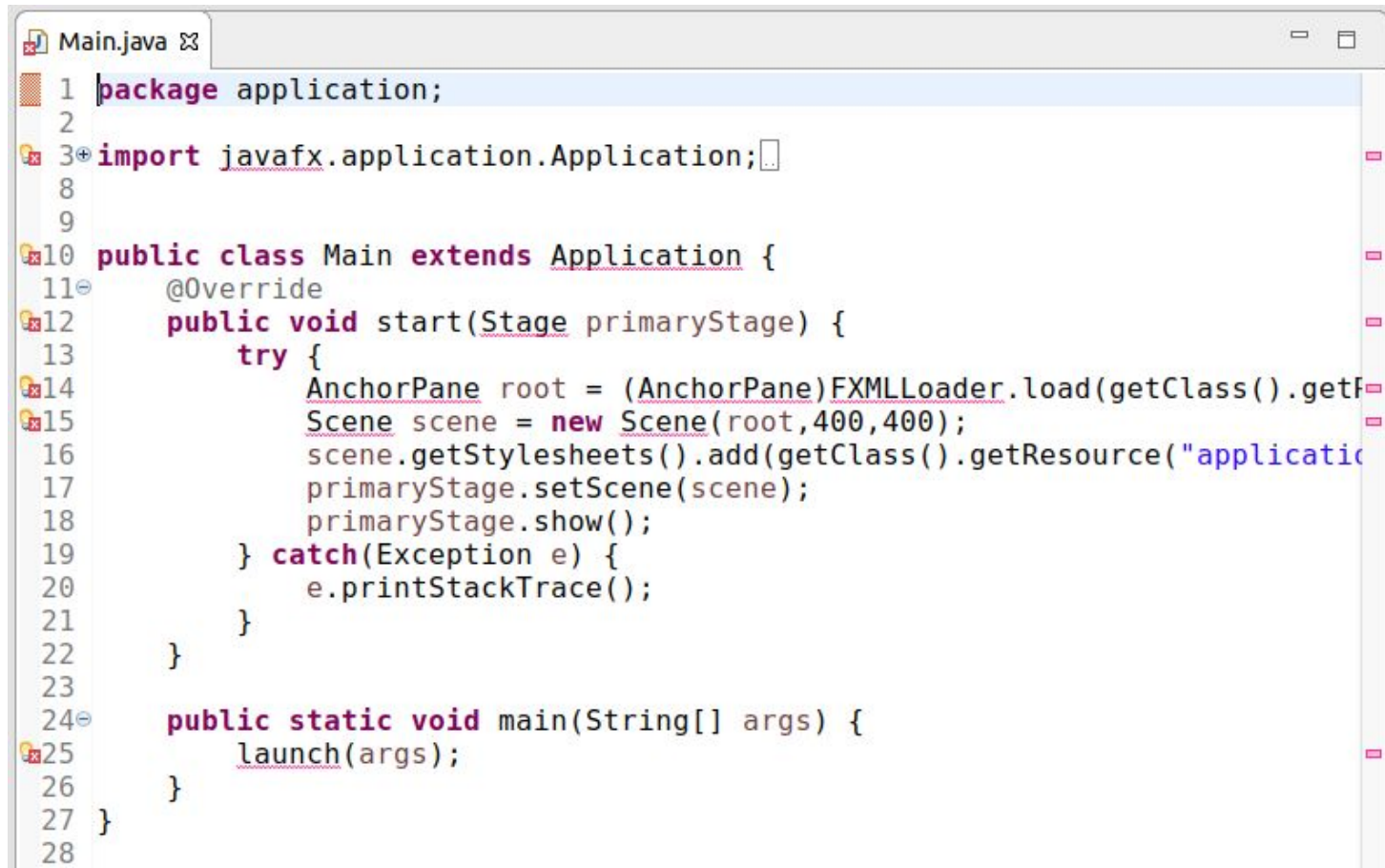
Ejercicio 1:

Añade la instrucción `primaryStage.setTitle("Hola (Tu Nombre)!")` justo antes de la instrucción `primaryStage.setScene(scene)` y ejecuta el programa.

Posibles errores y las soluciones...

Error #1

✓ No compila el Proyecto



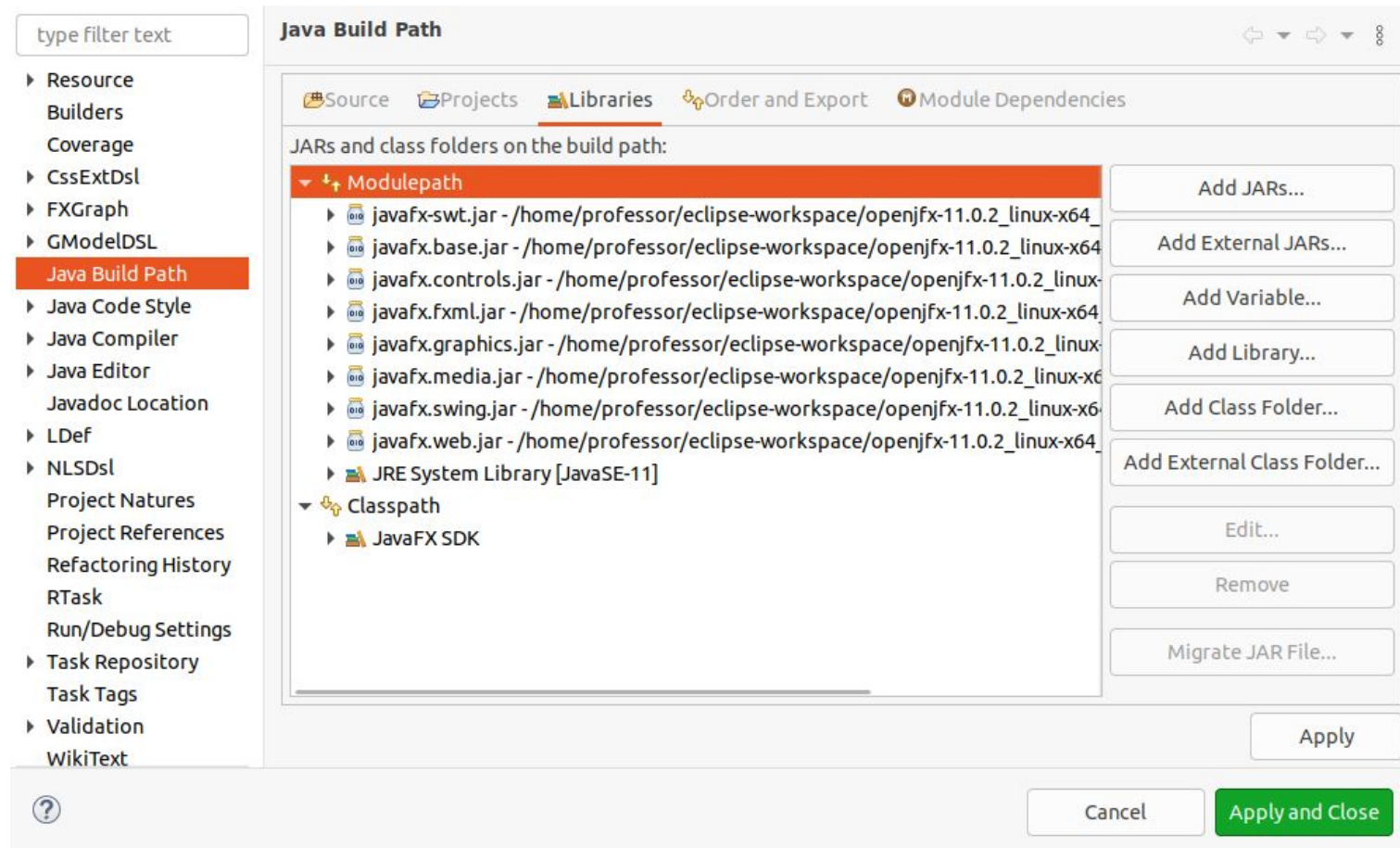
The screenshot shows a Java IDE window titled 'Main.java'. The code is as follows:

```
1 package application;
2
3 import javafx.application.Application;
4
5
6
7
8
9
10 public class Main extends Application {
11     @Override
12     public void start(Stage primaryStage) {
13         try {
14             AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("application.fxml"));
15             Scene scene = new Scene(root, 400, 400);
16             scene.getStylesheets().add(getClass().getResource("application.css"));
17             primaryStage.setScene(scene);
18             primaryStage.show();
19         } catch (Exception e) {
20             e.printStackTrace();
21         }
22     }
23
24     public static void main(String[] args) {
25         launch(args);
26     }
27 }
28
```

There are several red error icons in the left margin. The first is on line 3, indicating an 'import' error. The second is on line 14, indicating a 'getResource' error. The third is on line 15, indicating a 'new' error. The fourth is on line 16, indicating a 'getResource' error. The fifth is on line 25, indicating a 'launch' error.

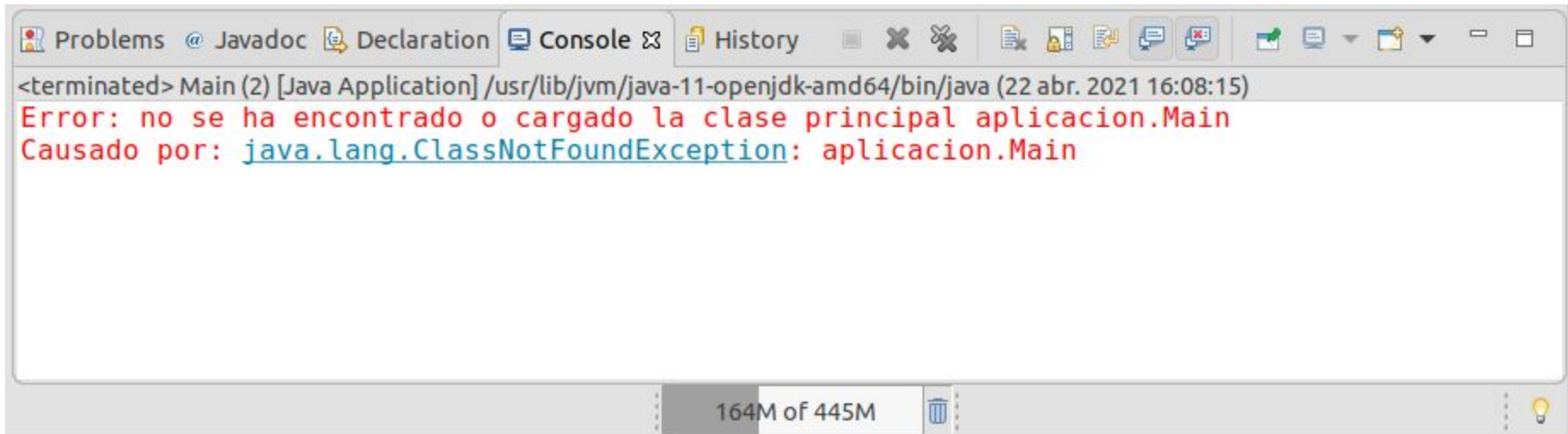
Solución #1

- ✓ Configurar Build Path añadiendo los .jar de JavaFX que se encuentran en la carpeta descomprimida del SDK.



Error #2

✓ Error de ejecución: **ClassNotFoundException**



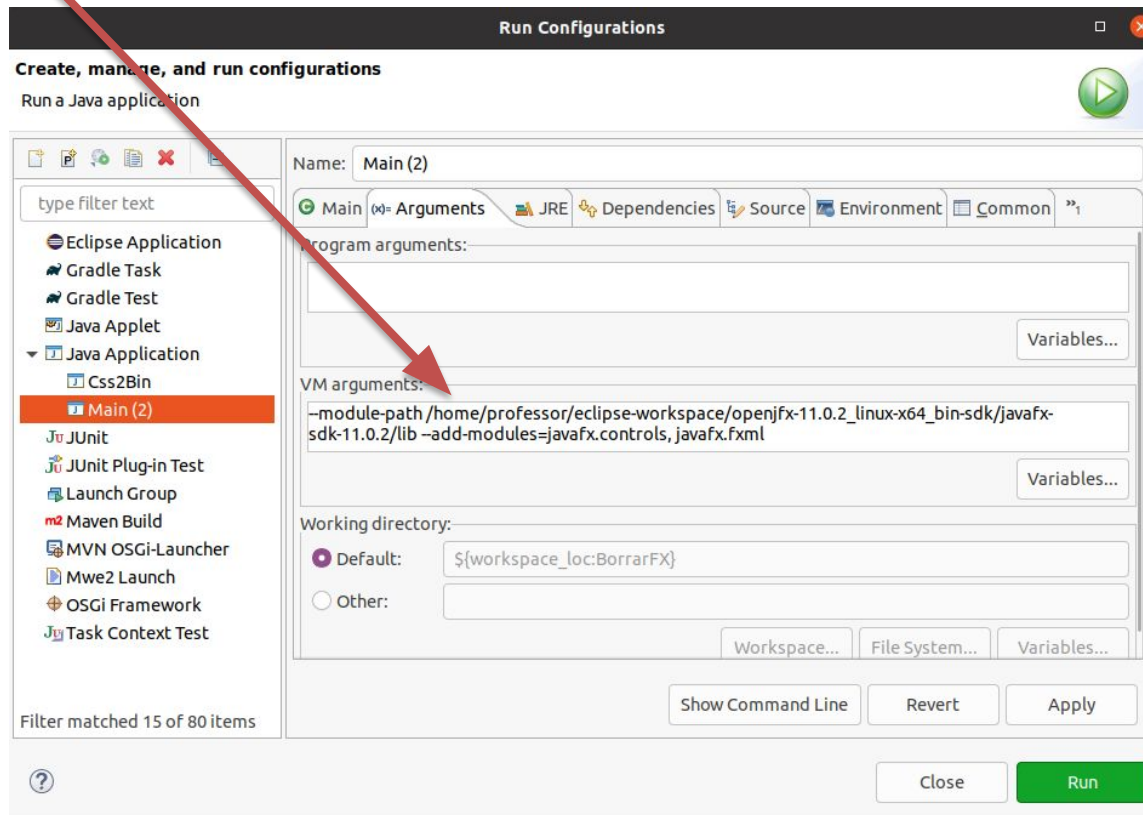
The screenshot shows an IDE's console window with the following content:

```
<terminated> Main (2) [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (22 abr. 2021 16:08:15)  
Error: no se ha encontrado o cargado la clase principal aplicacion.Main  
Causado por: java.lang.ClassNotFoundException: aplicacion.Main
```

The console window has a toolbar at the top with icons for Problems, Javadoc, Declaration, Console, History, and other IDE functions. At the bottom, there is a memory usage indicator showing '164M of 445M' and a trash icon.

Solución #2

- ✓ Añadir argumentos VM para la ejecución del proyecto.
- ✓ `--module-path "C:\openjfx-20.0.1_windows-x64_bin-sdk\javafx-sdk-20.0.1\lib"`
`--add-modules javafx.controls,javafx.fxml`



Entre las comillas debéis poner vuestro PATH al /lib del sdk de javafx que habéis descargado y descomprimido.

Error #3

✓ Error de ejecución: **InvocationTargetException**

Solución #3

✓ En *main.java*

Sustituir:

```
// Cargar la ventana  
Parent root = FXMLLoader.load(getClass().getClassLoader().getResource(fxml));
```

Por:

```
// Cargar la ventana  
Parent root = FXMLLoader.load(getClass().getResource(fxml));
```

Error #4

- ✓ Error de ejecución: **NullPointerException**

Solución #4

- ✓ En *Calculadora.fxml*

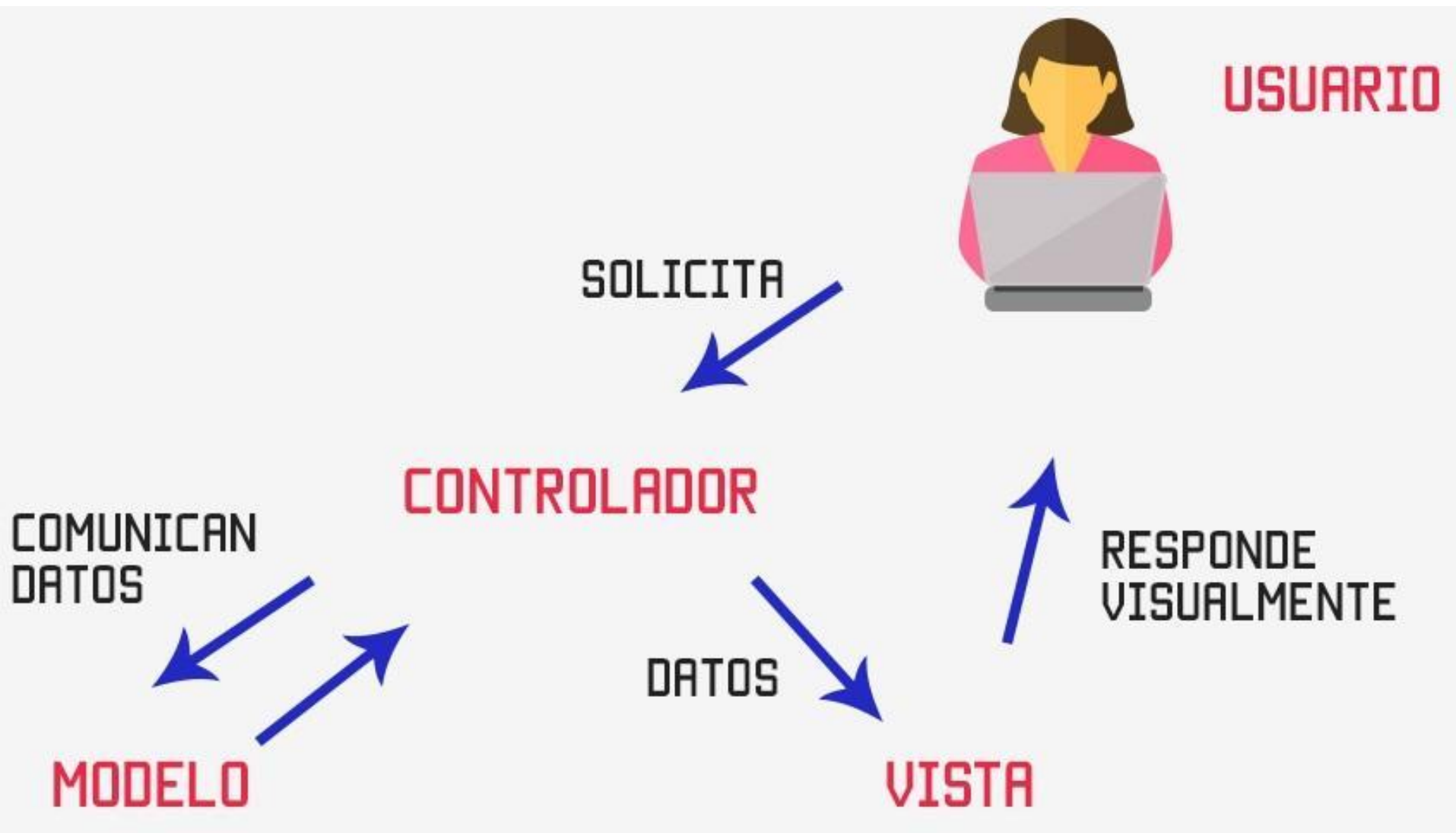
Revisar la ruta del controlador

- ✓ En *Main.java*

Revisar la ruta de la vista

Modelo – Vista – Controlador

Modelo – Vista – Controlador



Modelo – Vista – Controlador

El patrón **Modelo – Vista – Controlador (MVC)** es un patrón de diseño de software que se utiliza en muchos entornos distintos con muchos lenguajes de programación.

Se utiliza popularmente tanto para diseñar aplicaciones web y aplicaciones móviles, cómo aplicaciones de escritorio.

Si no utilizamos el patrón **MVC**, el código fuente de la **GUI** (*Graphical User Interface o Interfaz Gráfica de Usuario*) y los eventos y métodos con la funcionalidad de la interfaz, están en el mismo archivo, lo que hace que éste tenga un mayor número de líneas y sea muy difícil de modificar y de ampliar.

Si separamos en un archivo sólo el código para la **GUI** (*vista*), otro archivo sólo para controlar los eventos (*controlador*) y los archivos del dominio de nuestra aplicación (*modelo*), obtenemos un código mucho más simple, más fácil de comprender y mucho más fácil de mantener.

Esta es la filosofía principal del patrón **Modelo – Vista – Controlador**.

Modelo – Vista – Controlador

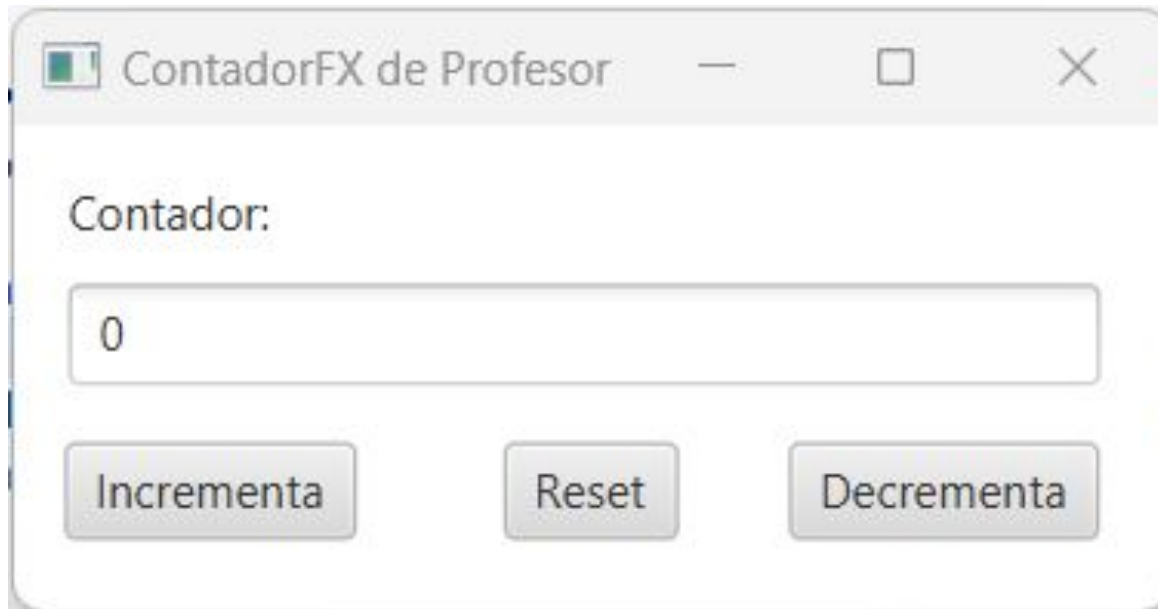
Los componentes de la arquitectura del patrón **MVC** están diseñados para manejar diferentes aspectos de una aplicación en desarrollo. El patrón de diseño MVC sirve para separar la capa de presentación de la lógica de la aplicación y es uno de los patrones de diseño de software más utilizados para el desarrollo web y de aplicaciones.

Este patrón de diseño separa los distintos aspectos de nuestro proyecto en 3 grupos:

- **Modelo:** Son todas las clases relacionadas con el dominio de nuestra aplicación. Clases que realizan la lógica del programa y clases que se utilizan para almacenar y gestionar los datos, a menudo, conectados con una base de datos
- **Vista:** Conocida como GUI o Interfaz Gráfica de Usuario. La vista contiene todas las funciones que interactúan directamente con el usuario, como hacer clic en un botón o un evento de entrada. Además se encarga de mostrar los datos almacenados al usuario.
- **Controlador:** El controlador conecta el *modelo* y la *vista*. Cuando un usuario interactúa con la IGU, solicita al controlador que se ejecute un evento. Este evento, va a comunicar con el modelo de la aplicación e intercambiar datos. Estos datos serán devueltos al controlador que los mostrará al usuario a través de la vista (IGU).

Modelo – Vista – Controlador

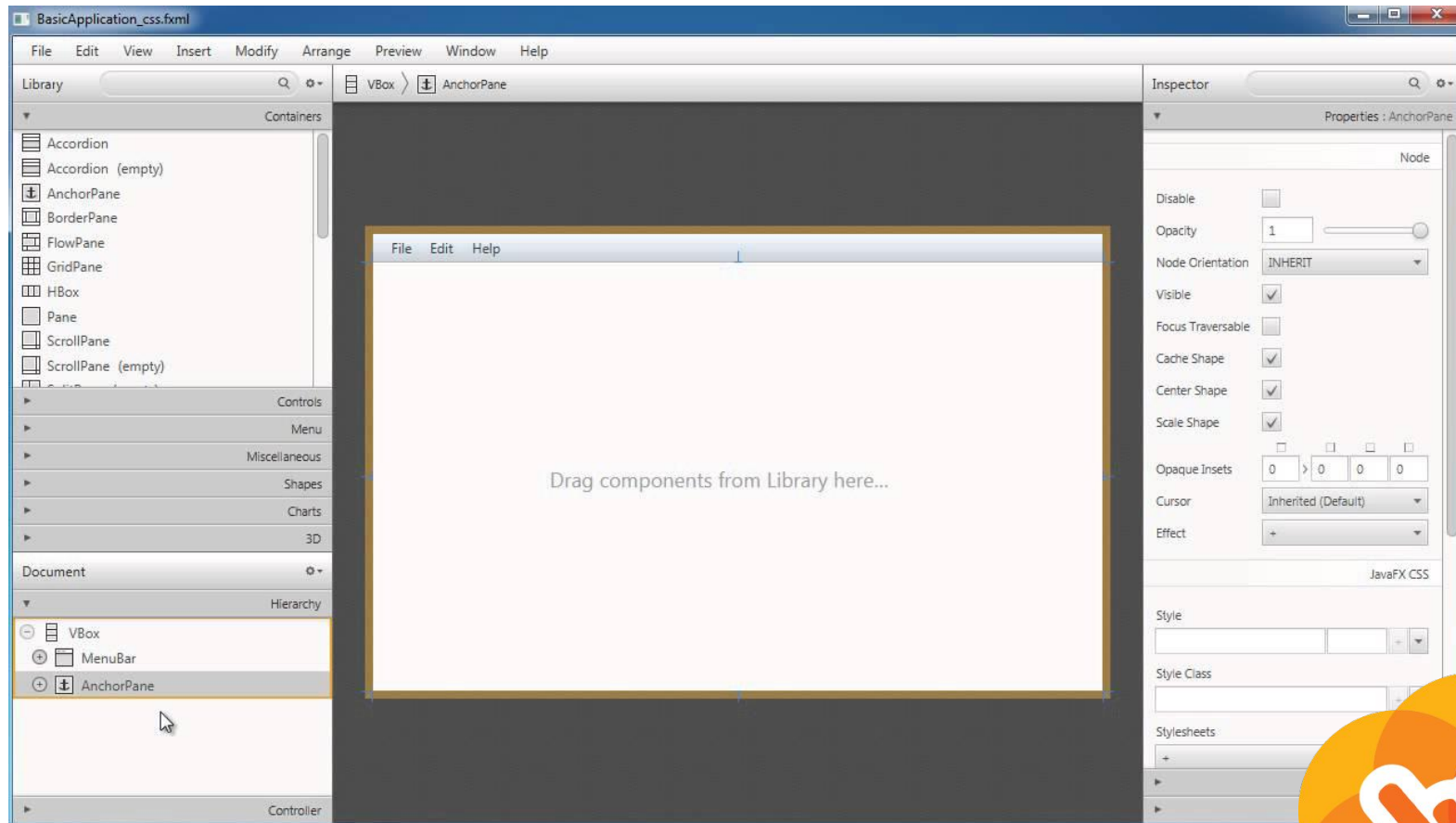
En los siguientes apartados seguiremos el ejemplo de realizar una aplicación sencilla de un contador:



Vista

Interfaz Gráfica de Usuario

Scene Builder



Stage – Scene - Nodes

En una aplicación JavaFX con una GUI, se diferencian tres grandes componentes:

- **Stage:** Es la ventana principal de la aplicación y viene determinada por el sistema operativo donde estamos ejecutando el programa. Una ventana de mismo programa se verá de forma diferente en cada S.O. sin que el programador pueda modificarlo.
- **Scene:** Es lo que se muestra dentro de la aplicación. La escena describe todo lo que hay dentro de una ventana en una aplicación **JavaFX**. La escena la define completamente el programador pudiendo diseñarla y cambiarla a su gusto.
- **Nodes:** Son todos aquellos componentes gráficos que conforman la escena. Estos nodos se almacenan en el fichero **FXML** en forma de árbol. Puedes repasar este concepto en el módulo de *Lenguajes de Marcas*.

Por tanto, tenemos que un **Stage** es el contenedor de nivel superior, que como mínimo consta de una escena y que a su vez es contenedora de otros componentes gráficos.

Resumiendo:

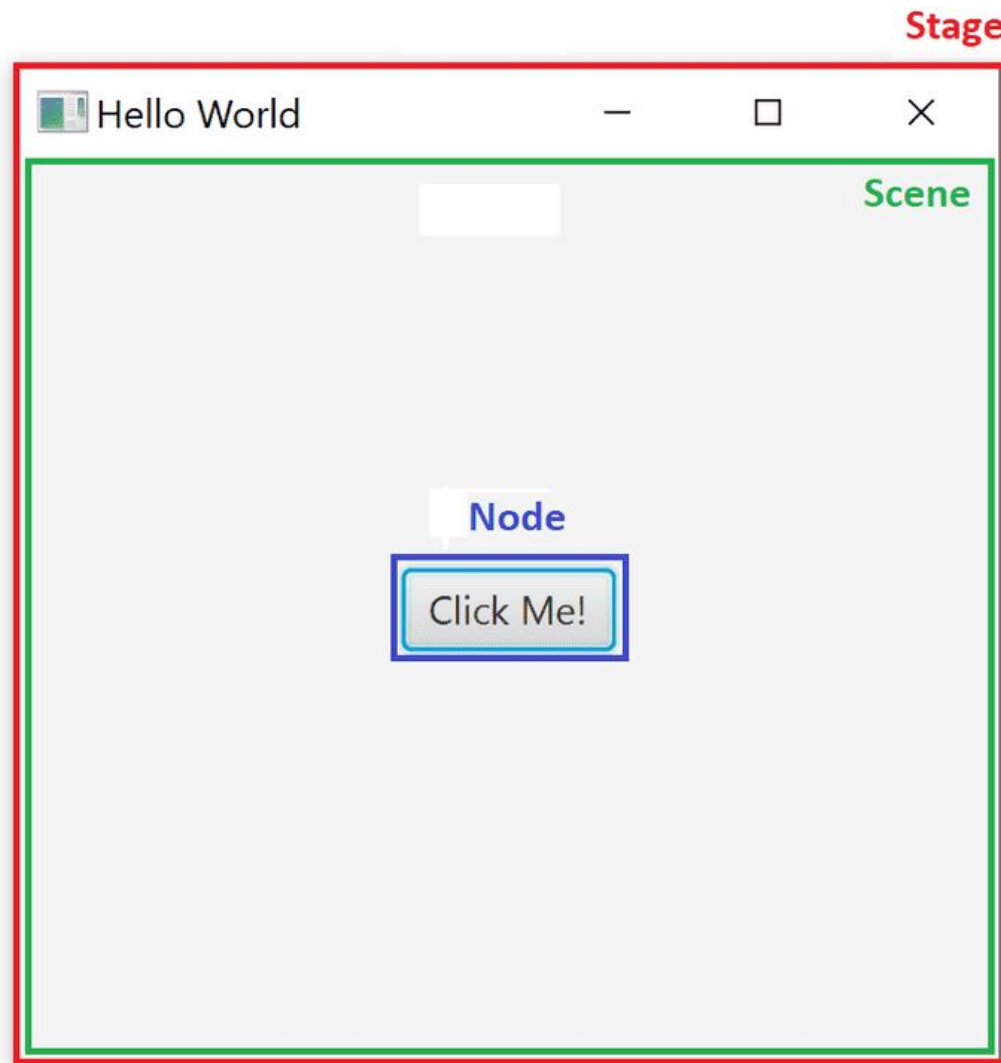
Stage : Se encarga del estilo y comportamiento de la GUI

Scene : Almacena todos los nodos en un nodo raíz y maneja los eventos

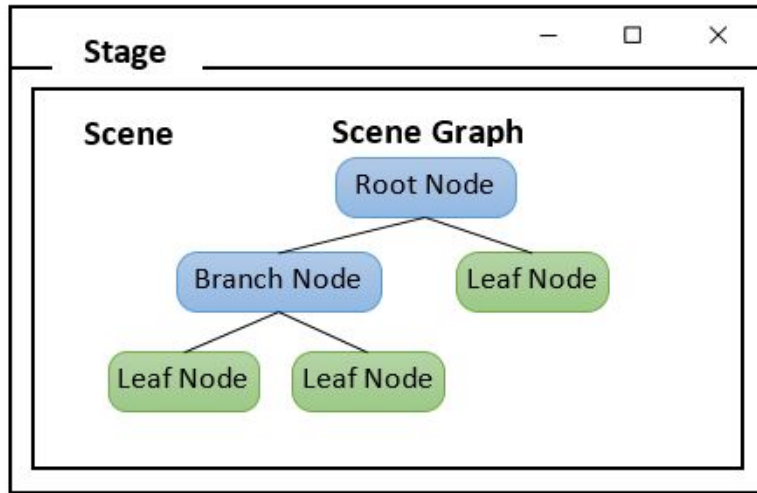
Nodes : Componentes o nodos que pertenecen a un Layout*.

(*) Los componentes de diseño (layout) en JavaFX **son los que nos permiten colocar de forma organizada el contenido de nuestra interfaz, ya sean botones, cajas de texto, imágenes, etc**

Stage – Scene - Nodes

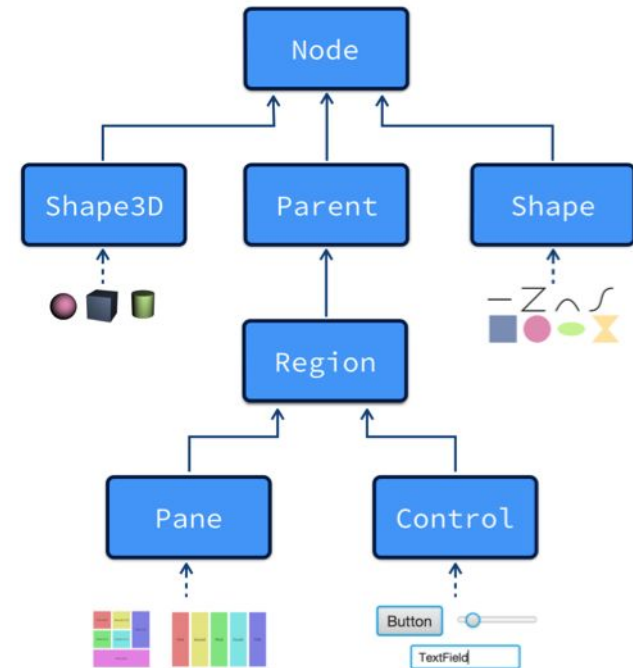
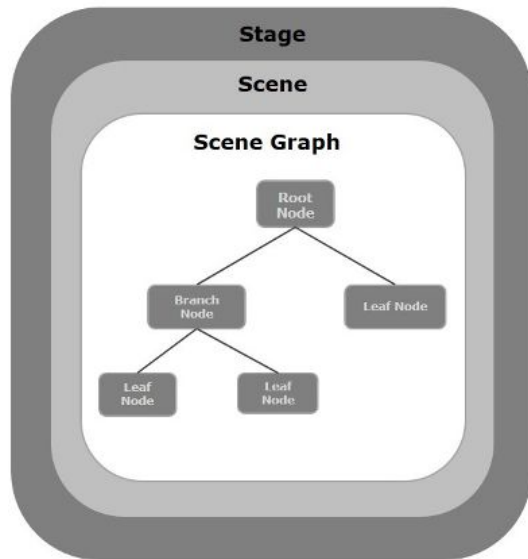


Stage – Scene - Nodes



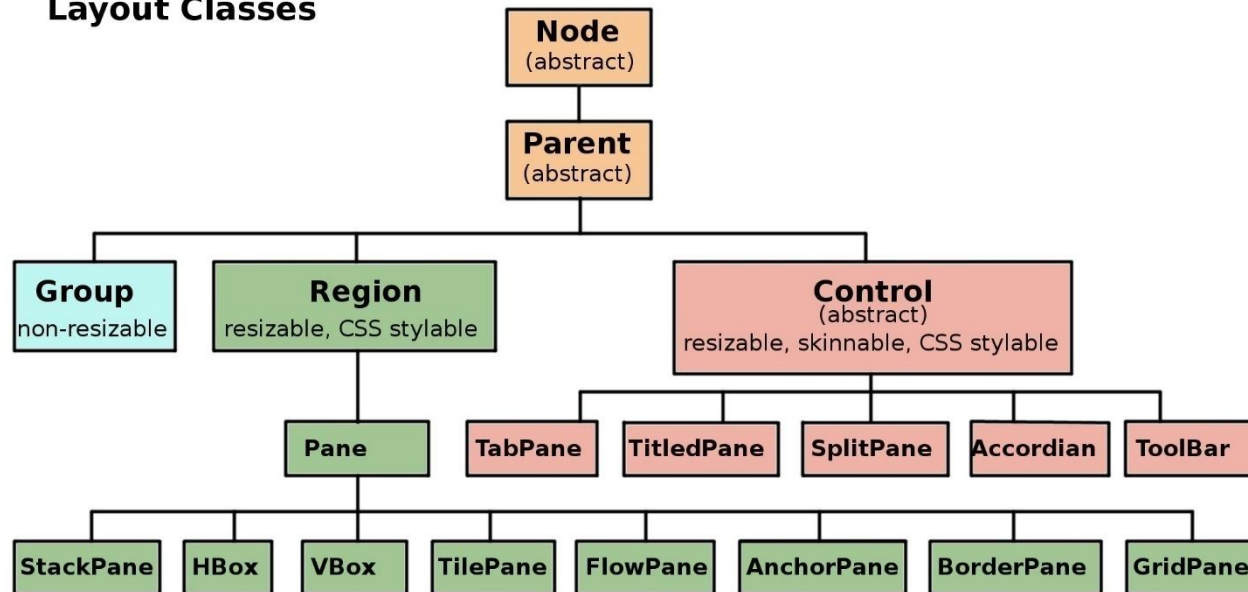
Decoration
(Title bar and border)

Content Area

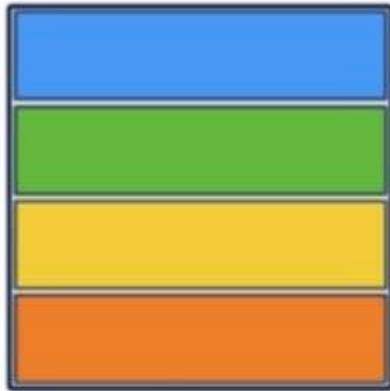


Layouts Classes

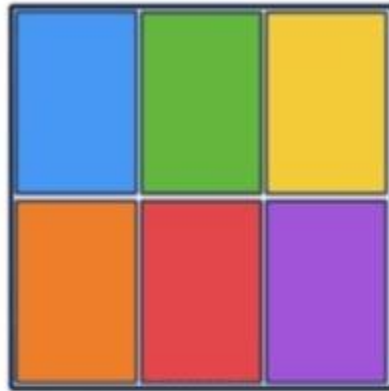
JavaFX Layout Classes



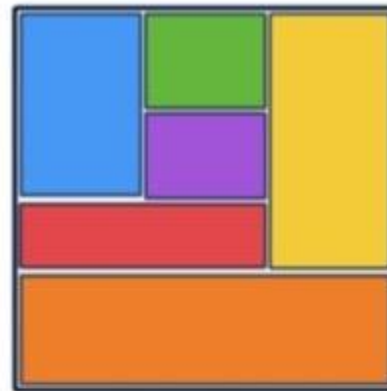
Layouts Pane



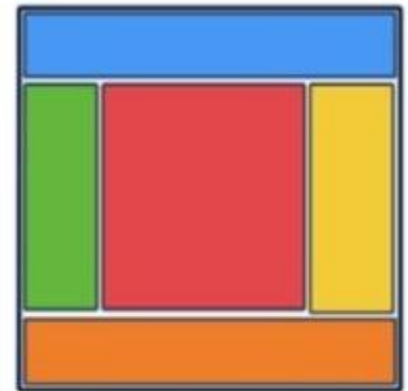
VBox



TilePane



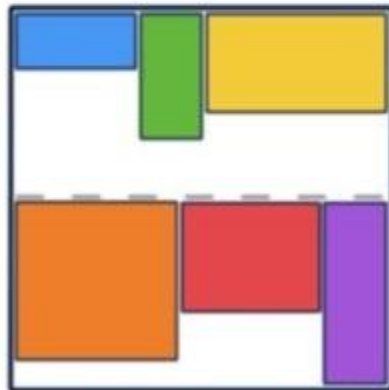
GridPane



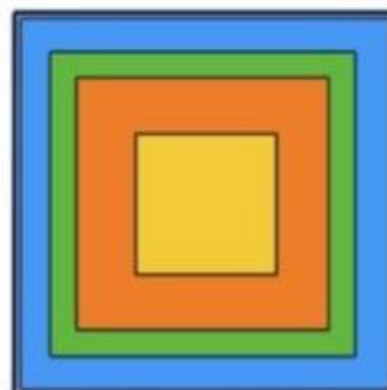
BorderPane



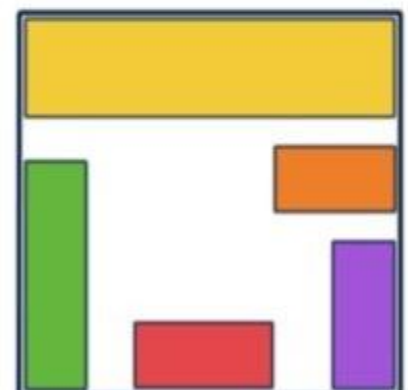
HBox



FlowPane



StackPane

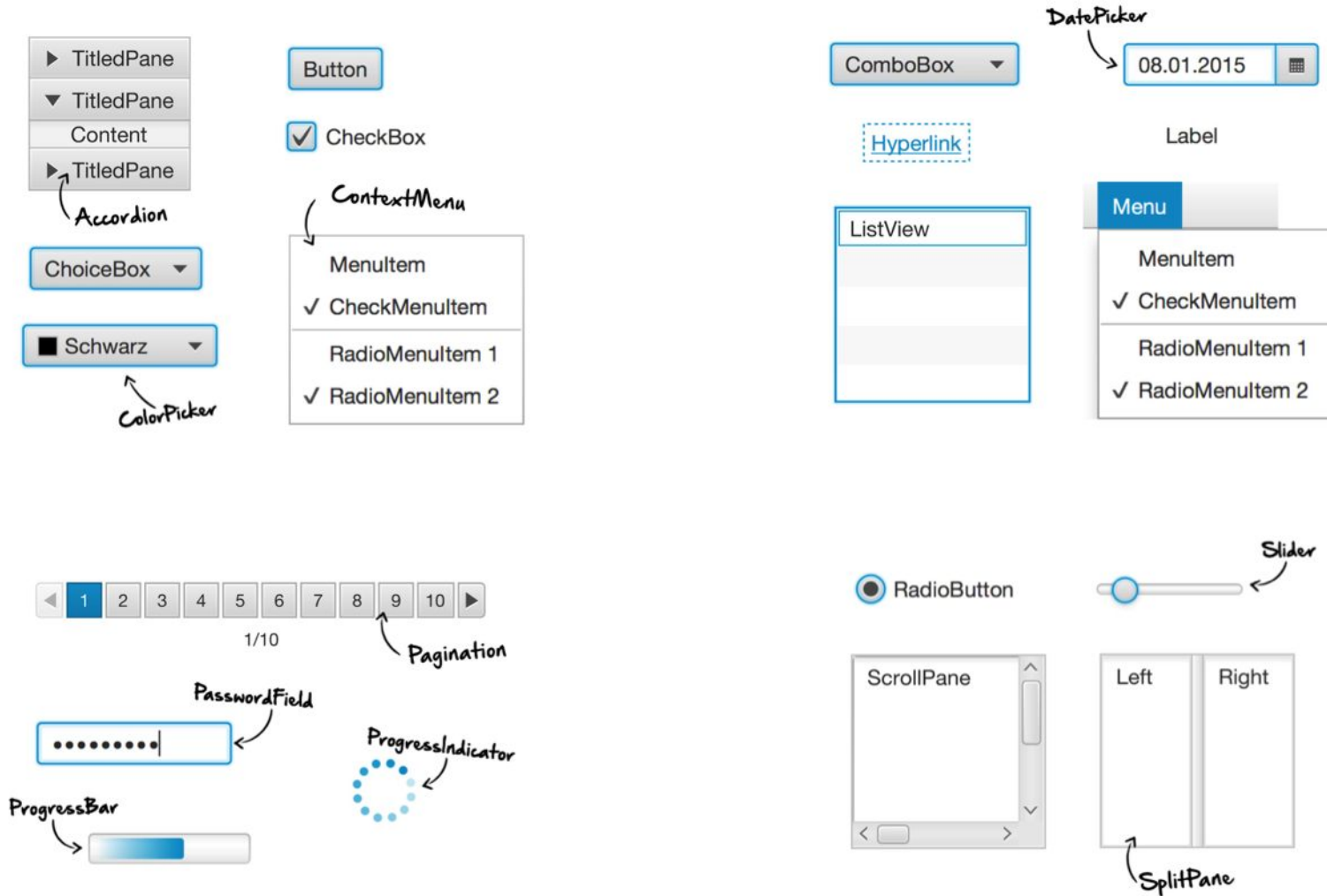


AnchorPane

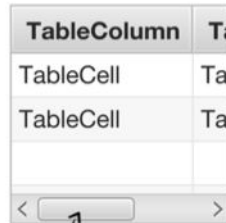
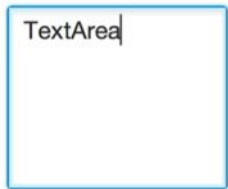
Layouts Pane

Panel	Descripción
VBox	Organiza de una forma sencilla una serie de nodos en una sola columna.
HBox	Organiza de una forma sencilla una serie de nodos en una sola fila.
TilePane	Un panel de mosaico es similar a un panel de flujo. El panel de diseño de TilePane coloca todos los nodos en una cuadrícula en la que cada celda o mosaico tiene el mismo tamaño. Los nodos se pueden colocar horizontalmente (en filas) o verticalmente (en columnas).
GridPane	Permite crear una cuadrícula flexible de filas y columnas en la que distribuir los nodos. Los nodos se pueden colocar en cualquier celda de la cuadrícula y pueden abarcar celdas según sea necesario. Es útil para crear formularios o cualquier diseño que esté organizado en filas y columnas.
BorderPane	Proporciona cinco regiones en las que colocar los nodos: superior, inferior, izquierda, derecha y central. Las regiones pueden ser de cualquier tamaño. Si su aplicación no necesita una de las regiones, no es necesario que la defina y no se le asigna espacio.
FlowPane	Los nodos se distribuyen consecutivamente y se ajustan al límite establecido para el panel. Los nodos pueden fluir verticalmente (en columnas) u horizontalmente (en filas).
StackPane	Organiza todos los nodos dentro de una sola pila con cada nodo nuevo agregado encima del nodo anterior. Este modelo de diseño proporciona una manera fácil de superponer texto en una forma o imagen o de superponer formas comunes para crear una forma compleja.
AnchorPane	Permite anclar nodos en la parte superior, inferior, izquierda, derecha o centro del panel. A medida que se cambia el tamaño de la ventana, los nodos mantienen su posición en relación con su punto de anclaje.

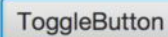
Componentes gráficos



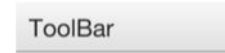
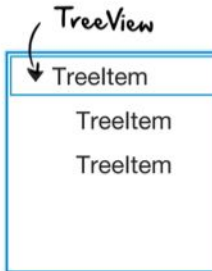
Componentes gráficos



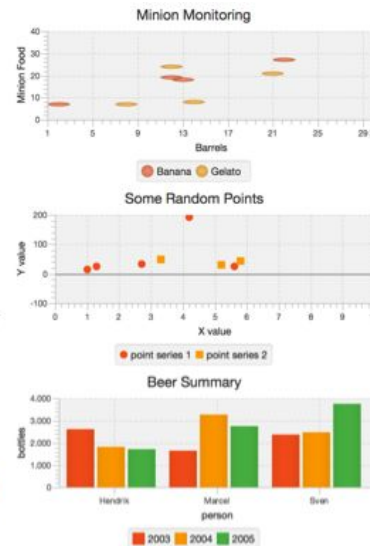
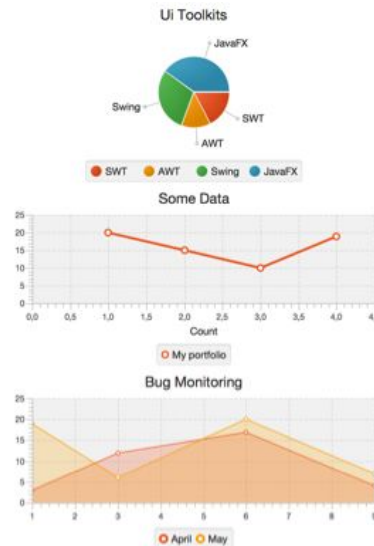
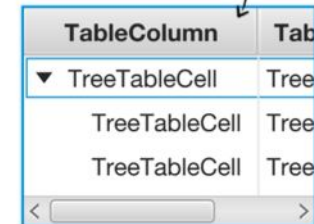
Table



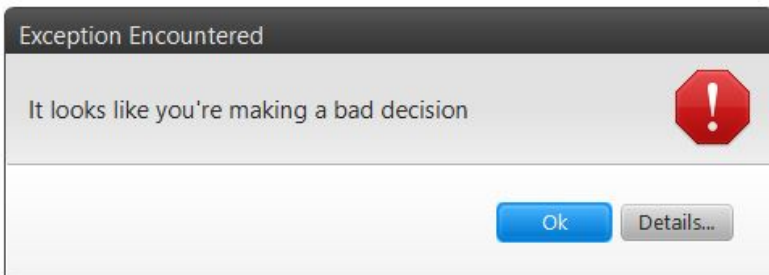
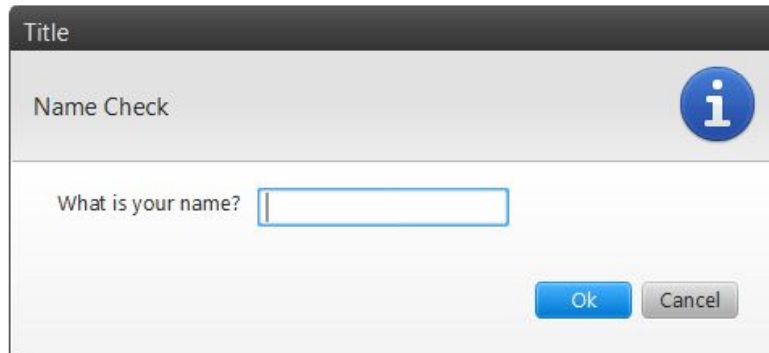
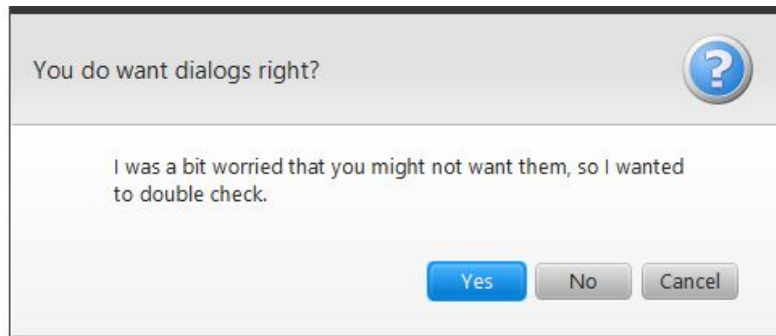
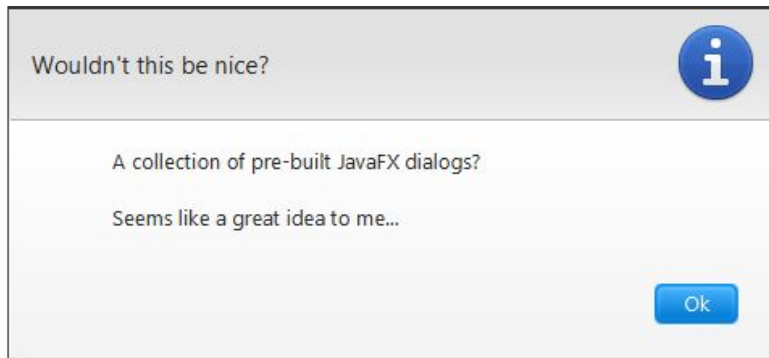
Some Node
with a tooltip



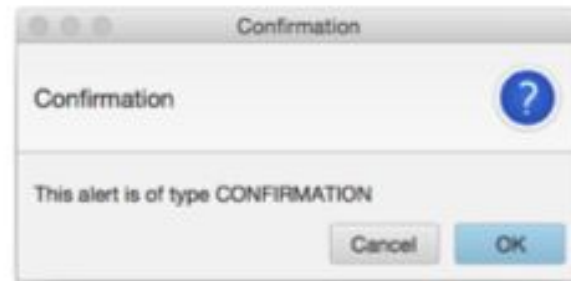
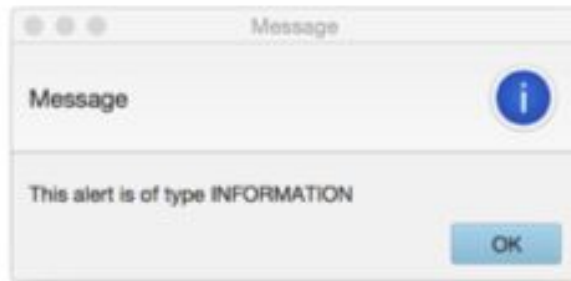
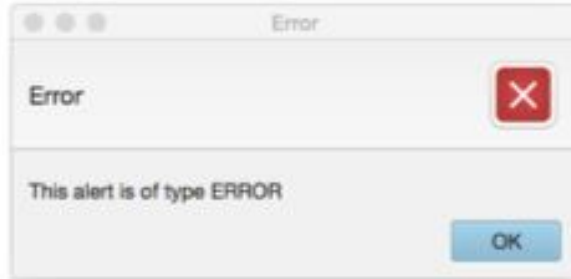
TreeTableView



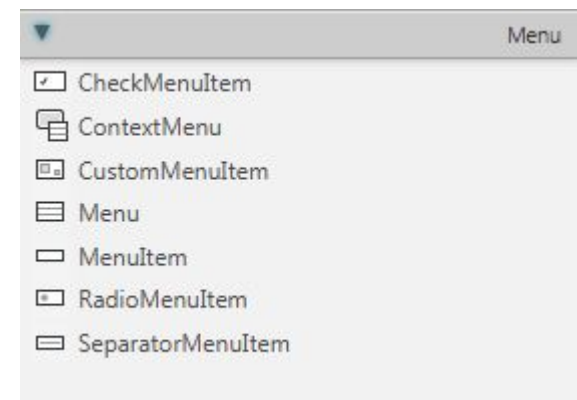
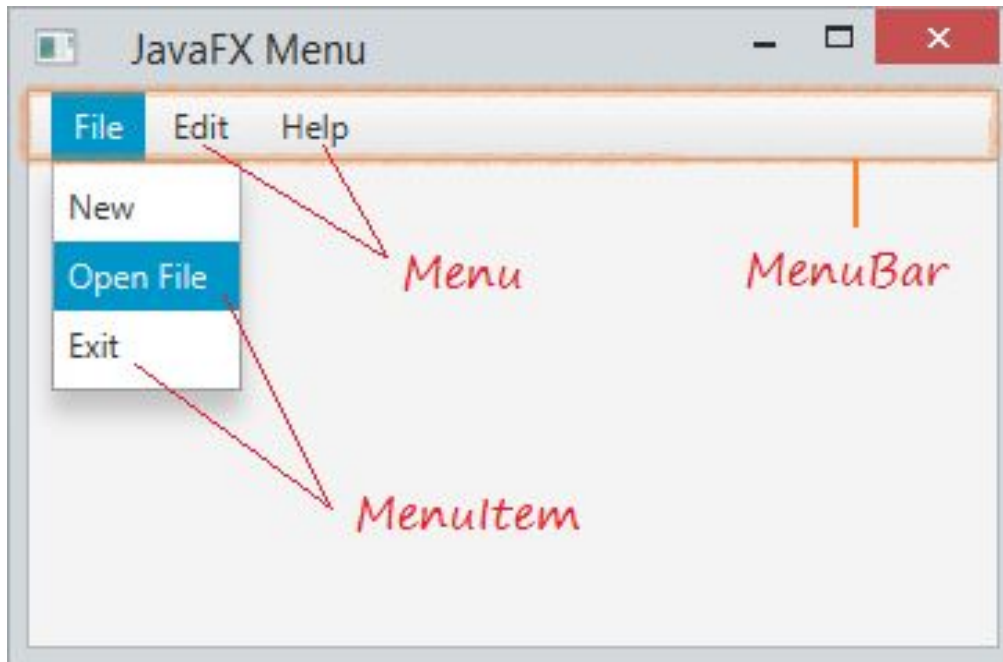
Dialogs and Alerts



Dialogs and Alerts



Menús



Menús

```
<MenuBar layoutX="0.0" layoutY="0.0" prefHeight="25.0" prefWidth="250.0">
  <menus>
    <Menu mnemonicParsing="false" text="Ver">
      <items>
        <MenuItem fx:id="menu_Historial" mnemonicParsing="false" text="Historial" />
        <MenuItem fx:id="menu_Salir" mnemonicParsing="false" text="Salir" />
      </items>
    </Menu>
    <Menu mnemonicParsing="false" text="Edición">
      <items>
        <MenuItem fx:id="menu_Copiar" mnemonicParsing="false" text="Copiar" />
        <MenuItem fx:id="menu_Pegar" mnemonicParsing="false" text="Pegar" />
      </items>
    </Menu>
    <Menu mnemonicParsing="false" text="Ayuda">
      <items>
        <MenuItem fx:id="menu_Ayuda" mnemonicParsing="false" text="Ver La Ayuda" />
        <MenuItem fx:id="menu_AcercaDe" mnemonicParsing="false" text="Acerca de Calculadora" />
      </items>
    </Menu>
  </menus>
</MenuBar>
```

Reutilizar un panel

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.control.ComboBox?>
4 <?import javafx.scene.control.Label?>
5 <?import javafx.scene.control.TextField?>
6 <?import javafx.scene.image.Image?>
7 <?import javafx.scene.image.ImageView?>
8 <?import javafx.scene.layout.AnchorPane?>
9
10 <AnchorPane prefHeight="741.0" prefWidth="575.0" stylesheets="@css/estilos.css" xmlns="http://javafx.com/javafx/11.0.1"
11     xmlns:fx="http://javafx.com/fxml/1" fx:controller="controlador.ElTiempoController">
12     <children>
13
14         <ImageView fitHeight="264.0" fitWidth="416.0" layoutX="85.0" layoutY="41.0" pickOnBounds="true" preserveRatio="true">
15             <image>
16                 <Image url="@img/aemet.png" />
17             </image>
18         </ImageView>
19         <TextField fx:id="txt_ciudad" layoutX="181.0" layoutY="280.0" onKeyReleased="#actualizarTiempo" prefHeight="25.0" prefWidth="292.0" />
20         <Label fx:id="label_mun" layoutX="97.0" layoutY="286.0" text="%lbl.municipio" />
21
22         <fx:include fx:id="panel_dia_1" prefHeight="300.0" prefWidth="150.0" source="PanelDia.fxml"
23             AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="50.0" AnchorPane.topAnchor="350.0" />
24         <fx:include fx:id="panel_dia_2" prefHeight="300.0" prefWidth="150.0" source="PanelDia.fxml"
25             AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="320.0" AnchorPane.topAnchor="350.0" />
26
27         <Label fx:id="label_idioma" layoutX="241.0" layoutY="686.0" prefHeight="25.0" prefWidth="174.0" text="%lbl.idioma" />
28         <ComboBox fx:id="combo_Idioma" layoutX="437.0" layoutY="680.0" onAction="#cambiarIdioma" prefWidth="150.0" />
29
30     </children>
31 </AnchorPane>
```

Controlador

Controlador

En el paquete **controlador**, la clase *ContadorController.java* implementa la interfaz *Initializable* y sobrescribe el método *initialize* de esta:

```
public class ContadorController implements Initializable{

    @FXML
    private TextField tfContador;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {

        // Crear la Calculadora
        contador = new Contador();
    }
}
```

En las clases XXXController se utiliza el método **initialize** que hace la función de constructor de clase. De esta forma este método podrá acceder a los atributos de la clase marcados con **@FXML**. El constructor por defecto de java no puede acceder a los atributos marcados con **@FXML**.

Los **atributos** marcados con **@FXML** son aquellos que están asociados a la Vista y por tanto les estamos diciendo que son accesibles desde el fichero de Vista correspondiente .fxml (por ejemplo, contador.fxml).

Eventos

En **JavaFX**, se produce un evento cada vez que el usuario interactúa con los componentes gráficos de la aplicación. Hay varias formas mediante las cuales el usuario puede generar el evento. Por ejemplo, el usuario puede hacer uso del ratón, puede presionar cualquier botón del teclado o puede minimizar o cerrar la ventana.

De ahí que podamos decir que los eventos son básicamente las notificaciones que nos indican que el usuario ha realizado alguna acción con nuestra aplicación.

Los principales tipos de eventos que se pueden gestionar con **JavaFX** son:

- **ActionEvent:** Un evento que representa algún tipo de acción. Es un evento genérico que luego se debe especificar que hace realmente. Este tipo de evento se usa ampliamente para representar una variedad de acciones.
- **InputEvent:** Un evento que indica una entrada de usuario. Como cuándo se ha pulsado un botón, se ha pulsado una tecla, se ha liberado la tecla pulsada y otros usos similares.
- **WindowEvent:** Evento relacionado con acciones de mostrar / ocultar ventanas.

Interacción con el usuario - Eventos

Opción A.- Asignar evento en la vista (.fxml)

Contador.fxml (código FXML):

```
<Button fx:id="btnIncrementa" layoutX="13.0" layoutY="82.0"
mnemonicParsing="false" onAction="#incrementa" text="Incrementa" />
```

ContadorController.java (código Java):

```
@FXML
private Button btnIncrementa;

// Event Listener on Button[#btnIncrementa].onAction
@FXML
public void incrementa(ActionEvent event) {
    // TODO Autogenerated
}
```

Interacción con el usuario - Eventos

Opción B.- Asignar eventos, desde el controlador, al inicializar la vista.

Contador.fxml

```
<Button fx:id="btnIncrementa" layoutX="13.0" layoutY="82.0"
mnemonicParsing="false" text="Incrementa" />
```

ContadorController.java

```
public class ContadorController implements Initializable {

    // Atributos graficos FXML
    @FXML
    private Button btnIncrementa;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        // Evento botones, apertura de ventanas
        btnIncrementa.setOnMouseClicked((event) -> incrementa());
    }
}
```

Interacción con el usuario - Eventos

(Sigue) Opción B.- Asignar eventos, desde el controlador, al inicializar la vista.

CalculadoraController.java

```
public class ContadorController implements Initializable {

    // Atributos graficos FXML
    @FXML
    private Button btnIncrementa;

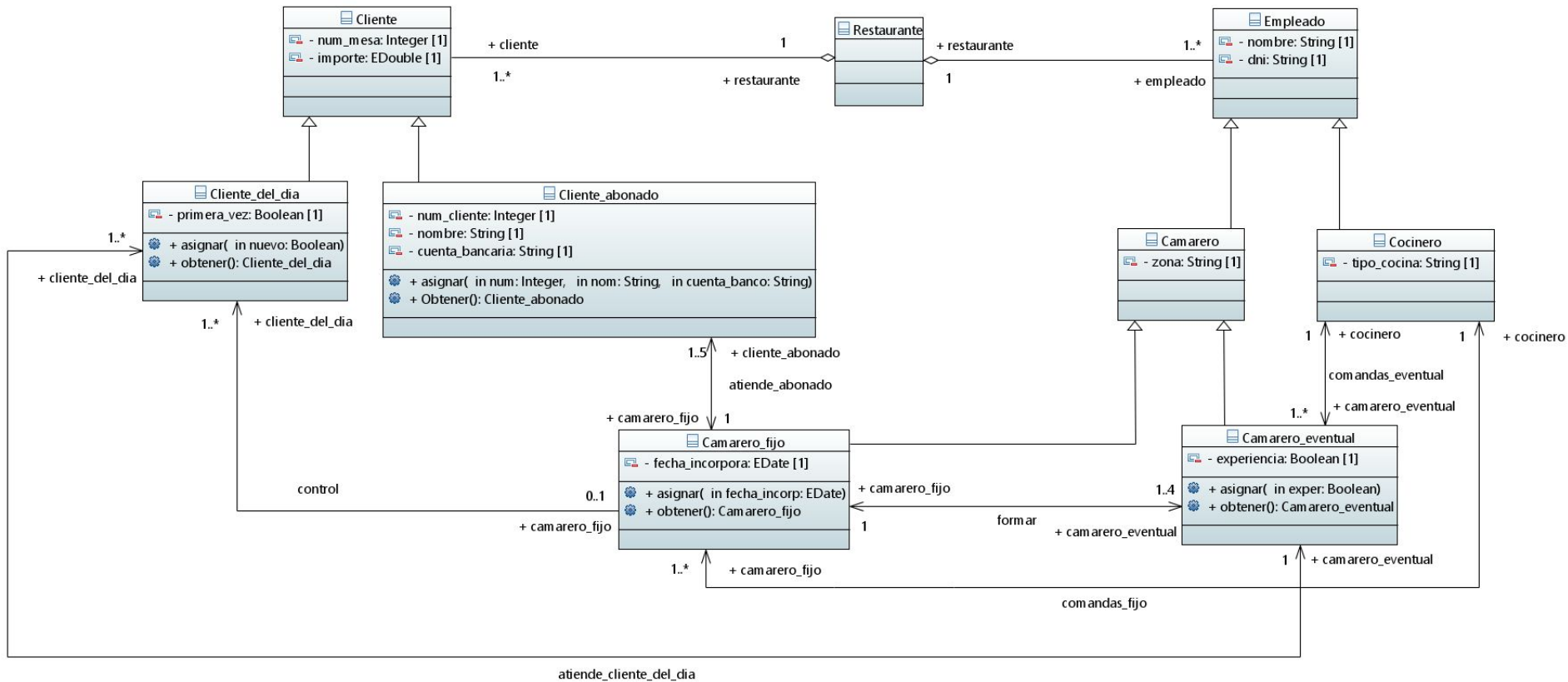
    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        // Evento botones, apertura de ventanas
        btnIncrementa.setOnMouseClicked((event) -> incrementa());
    }
}

// Ahora incrementa es un método "normal" y no necesita ni @FXML ni(ActionEvent)
public void incrementa() {
    // TODO Autogenerated
}
```

Modelo

Modelo

✓ Diagrama de clases



Modelo

Podemos crea un modelo de contador escribiendo una clase llamada Contador.java:

```
public class Contador {  
    private int numero;  
  
    public Contador() {  
        this.numero = 0;  
    }
```

```
// ... Resto de métodos que modelan el comportamiento de un  
contador
```

```
}
```

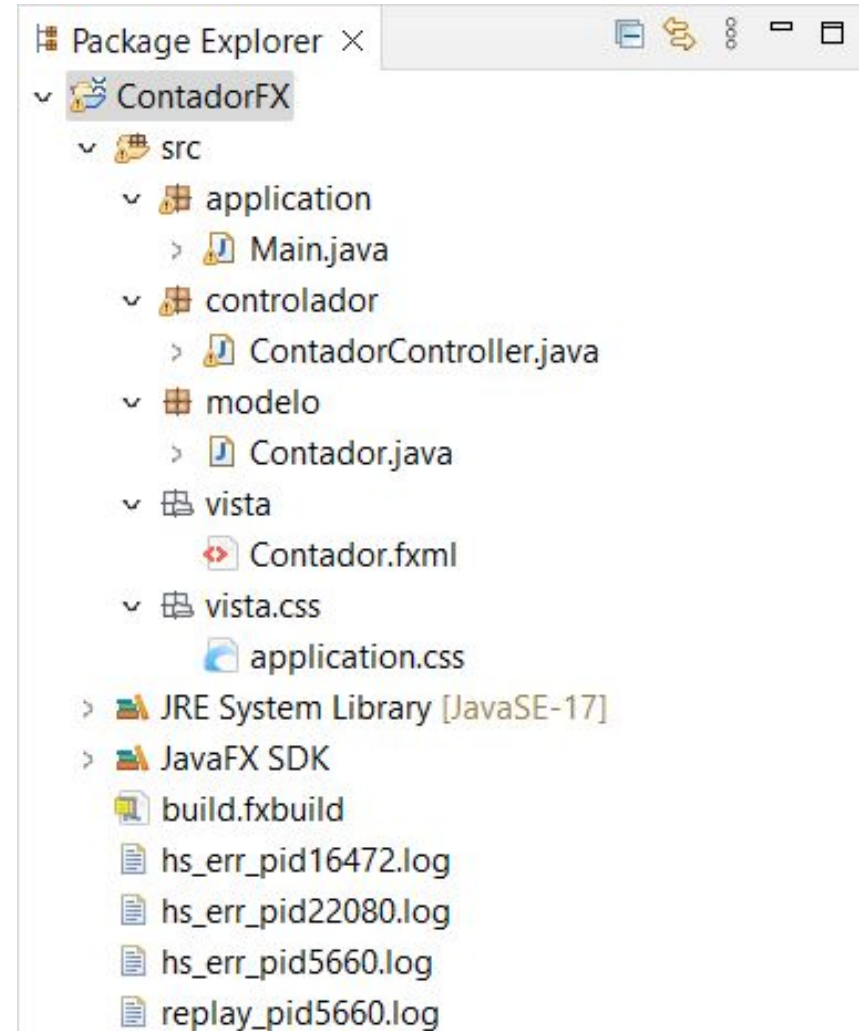
Organización del proyecto MVC

Organización del proyecto MVC

El proyecto lo podemos organizar en paquetes siguiendo el concepto del patrón MVC:

- **Paquete application:** donde se encuentra el *Main.java*, que es la clase de entrada a nuestra aplicación.
- **Paquete controlador:** contiene el controlador *XXXController.java* correspondiente a la vista *XXX*.
- **Paquete modelo:** contiene el fichero *XXX.java* que corresponderá al modelo de datos.
- **Paquete vista:** contiene el *XXX.fxml* de la vista. Atención, los recursos asociados a la vista se pueden situar en paquetes *vista.css*, *vista.imágenes*, etc.

Para la mayoría de aplicaciones a pequeña escala normalmente tendremos una vista y un controlador asociado. Sin embargo, en GUIs más complejas podríamos tener distintas vistas con controladores correspondientes.



Organización del proyecto MVC

Para que no de problemas la compilación de nuestra aplicación al crear la estructura de paquetes, debemos modificar la instrucción de carga del FXMLLoader en el Main.java:

- Para el Layout (AnchorPane, Parent, etc) añadir el método getClassLoader() a la siguiente instrucción:

```
AnchorPane root = (AnchorPane)FXMLLoader.Load(getClass().getResource("vista/Contador.fxml"));
```

Y quedaría así:

```
AnchorPane root = (AnchorPane)FXMLLoader.Load(getClass().getClassLoader().getResource("vista/Contador.fxml"));
```

- Para la ruta del css, cambiar la ruta según el nombre del paquete. Por ejemplo, si el fichero .css está en el paquete vista.css entonces la ruta es /vista/css/micss.css (date cuenta que hay que poner la barra "/" al inicio en este caso):

```
scene.getStylesheets().add(getClass().getResource("/vista/css/application.css").toExternalForm());
```

Personalizando la aplicación

Icono de la aplicación

Main.java

```
@Override
public void start(Stage primaryStage) {

    ...

    // Asignar icono de la aplicación
    primaryStage.getIcons().add(new
    Image(getClass().getResource("/vista/img/icon.png").toExternalForm()));

    ...

}
```


Mostrar otra ventana

```
//Mostrar otra ventana
private void mostrarVentanaAyuda(String rutaFXML, String titulo) {

    try{

        //Léeme el source del archivo que te digo fxml y te pongo el path
        FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(rutaFXML));
        Parent root = (Parent) fxmlLoader.load();

        //Creame un nuevo Stage (una nueva ventana vacía)
        Stage stage = new Stage();

        //Asignar al Stage la escena que anteriormente hemos leído y guardado en root
        stage.setTitle(titulo);
        stage.setResizable(false);
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setScene(new Scene(root));

        //Mostrar el Stage (ventana)
        stage.show();

    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

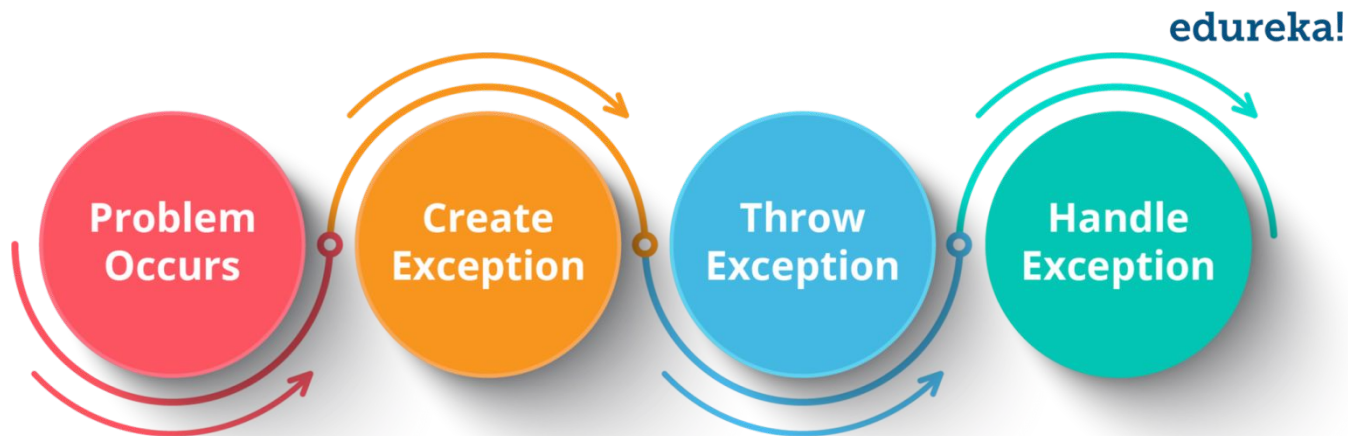
El toque profesional...

Excepciones

En el paquete **Excepciones** crear, al menos, las siguientes excepciones:

`NumeroNegativoException.java`

Añadir la funcionalidad necesaria en la aplicación utilizando las excepciones creadas.



Estilos CSS

Las hojas de estilo en cascada de JavaFX se basan en las reglas de W3C CSS. El objetivo de JavaFX CSS es permitir que los desarrolladores web que ya están familiarizados con CSS para HTML utilicen CSS para personalizar y desarrollar estilos para controles JavaFX y objetos de gráficos de escena de forma natural.

Todos los conceptos CSS que has estudiado en el módulo de Lenguaje de Marcas, son válidos para el diseño de IGU con JavaFX.

En la medida de lo posible, JavaFX CSS sigue los estándares de W3C; sin embargo, los nombres de propiedad de JavaFX han sido prefijados con una extensión "-fx-".



Estilos CSS

Main.java

```
@Override
public void start(Stage primaryStage) {

    ...

    // Asignar hoja de estilos
    scene.getStylesheets().add("/vista/css/estilos.css");

    ...

}
```

Estilos.css

```
.button {
    -fx-background-color: #FBFCFC;
    -fx-text-fill: #979A9A;
}
```

KeyListener

//1.- Asignar evento al TextArea

```
<TextArea fx:id="display" editable="false" layoutX="10.0" layoutY="37.0"
onKeyPressed="#pulsarTecla" prefHeight="100.0" prefWidth="245.0" text="0" />
```

//2.- Desactivar que puedan quedarse el foco de la aplicación TODOS los botones

```
<Button fx:id="boton_0" focusTraversable="false" layoutX="10.0" layoutY="400.0"
mnemonicParsing="false" prefHeight="45.0" prefWidth="95.0" text="0" />
```

//3.- Programar evento en el controlador

@FXML

```
void pulsarTecla(KeyEvent event) {

    switch (event.getCode()) {

        case DIGIT0: case NUMPAD0: insertarNumero("0"); break;
    }
}
```

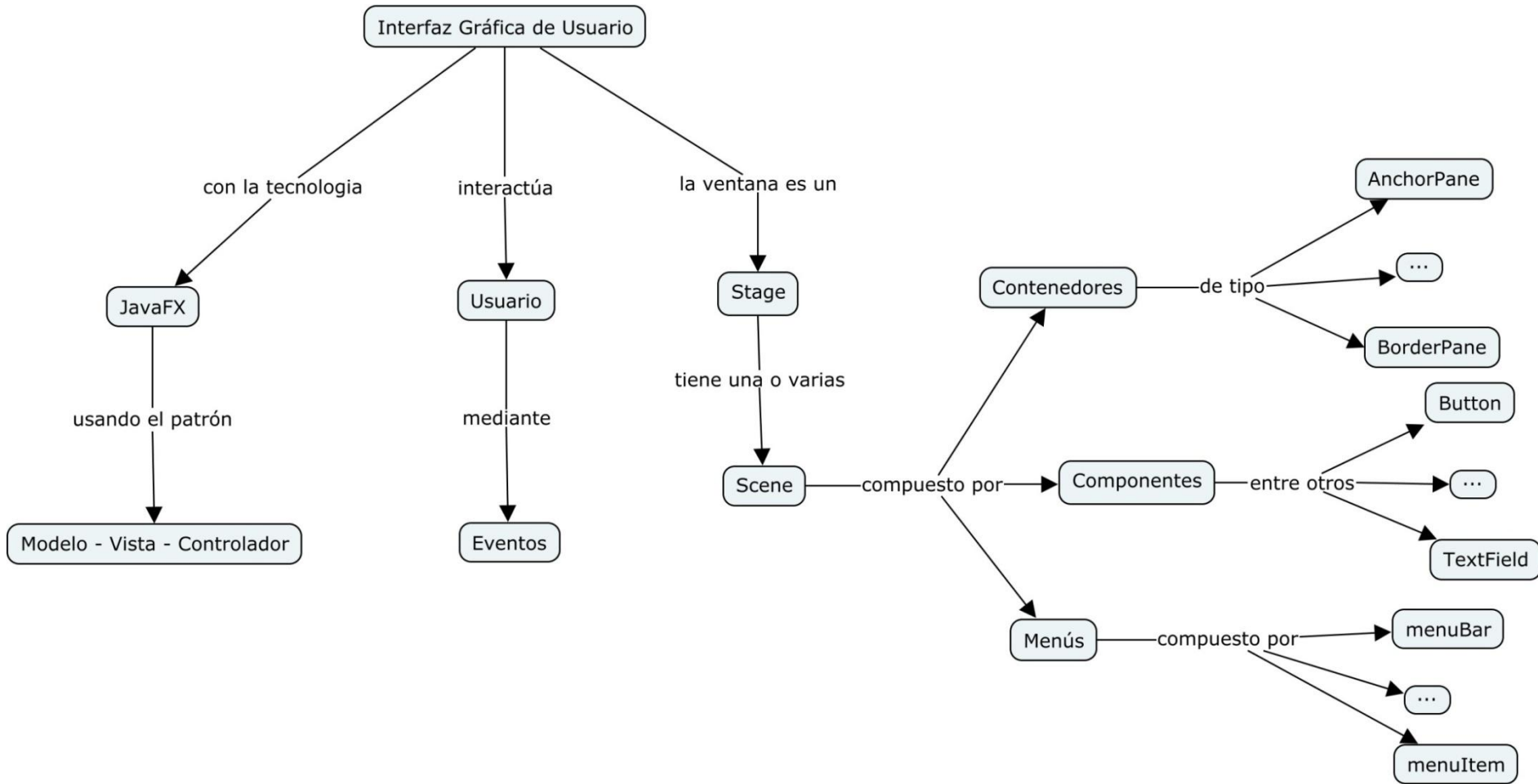
//4.- Combinaciones de teclas

//MC

```
KeyCombination ctrlL = new KeyCodeCombination(KeyCode.L, KeyCodeCombination.CONTROL_DOWN);
if (ctrlL.match(event)) memoryClear();
```

Resumen

Resumen



Para terminar...

Despliegue de la aplicación (.jar)

Para exportar nuestro proyecto a una aplicación real debemos exportarlo a **.jar**.

Click derecho sobre el proyecto, **exportar** y escogemos la opción **runnable JAR file**.

En library handling escogeremos la segunda opción. Esta opción empaqueta las librerías necesarias dentro del **.jar**.

Este .jar podrá ejecutarse por consola con el comando:

```
java -jar ContadorFX.jar
```

También se puede hacer doble clic sobre el ejecutable o crear un acceso directo.

Existen programas para exportar el **.jar** a **.exe** si se considera oportuno.