DESARROLLO DE INTERFACES
2º DAM

I.E.S. POLITÉCNICO H.LANZ
JOSÉ MARÍA MOLINA

TEMA 2-1-2 – COMPONENTES JAVAFX

2-1-2 COMPONENTES JAVAFX

Vamos a crear GUIs mediante JAVAFX. Esta tecnología permite crear la GUI por código o de forma gráfica mediante ficheros XML(Interfaz) creando así una estructura MVC, donde la Vista se **puede** separar por completo del Modelo y Controlador.

La funcionalidad siempre habrá que hacerla por código…

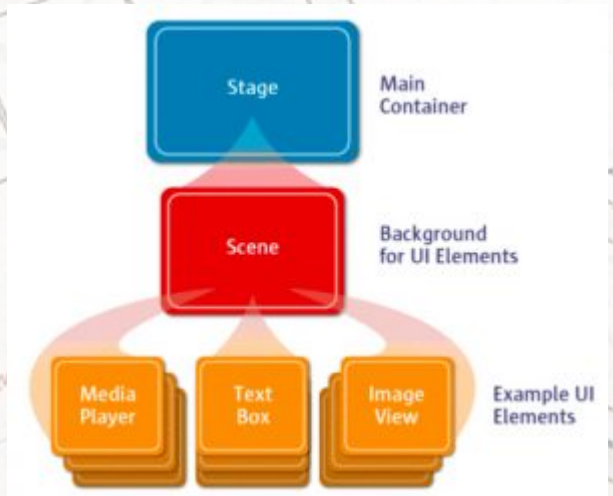
MUY IMPORTANTE: Los apuntes más técnicos son el propio código de los ejemplos, de otra forma unos hipotéticos apuntes serían demasiado extensos. Además, ver un componente funcionando facilita su entendimiento

- ✓ 1 FUNDAMENTOS Y COMPONENTES
- ✓ 2 EJEMPLOS
- ✓ 3 AÑADIR RECURSOS
- ✓ 4 ANEXO

1 FUNDAMENTOS Y COMPONENTES

JERARQUÍA DE ÁRBOL Y MVC

- **Stage**: es el contenedor principal de la aplicación. Viene a ser el marco de una "ventana". Puede haber varios Stages en la App.
- **Scene**: Cada Stage tiene asociado un Scene, siendo el contenedor de todos los elementos gráficos de la ventana. Se puede cambiar el Scene del Stage (estamos "cambiando contenido de ventana")
- **UI Components**: Son cada uno de los elementos gráficos que se representarán dentro de una escena



1 FUNDAMENTOS Y COMPONENTES

JERARQUÍA DE ÁRBOL Y MVC

- **JavaFX → Estructura de Árbol:** permite gestionar los elementos que conforman la interfaz de una forma dinámica
  - **Raíz (parent)**: es de tipo Contenedor y solo puede haber uno por Scene, es el contenedor principal del que cuelga todo. Ej: (VBox, HBox, StackPane, AnchorPane, etc..)
  - **Nodo (node)**: puede ser de cualquier tipo de componente: contenedor, nodos hoja, gráficos, etc...
  - **Hoja**: son **Nodos** que nunca van a tener descendencia (Label, TextField, Label, Button, ImageView, etc)
- **Métodos útiles**: nodo.getChildren().add(nodo) añade un nodo nuevo y nodo.getChildren().addAll(nodo1,nodo2,…) añade muchos nodos de golpe

1 FUNDAMENTOS Y COMPONENTES

JERARQUÍA DE ÁRBOL Y MVC

- **Clase Principal**: es el punto de inicio del programa, hereda de **Application** y su constructor llama al método launch (importante: solo se puede llamar a un launch por aplicación)
- El método **main** NO SE TOCA.
- A su vez, el launch llama al método start **donde :**
  - Se crea (como mínimo) un nodo raíz del árbol (contenedor) y los componentes (nodos/hojas)
  - Se crea una escena (Scene) donde están los componentes
  - Se asigna el Scene al Stage y se muestra

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

/**
 * Plantilla JAVAFX
 */
public class Plantilla extends Application {

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

    @Override
    public void start(Stage primaryStage) {

        VBox vbox = new VBox();
        Scene scene = new Scene(parent: vbox);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

}
```

1 FUNDAMENTOS Y COMPONENTES

JERARQUÍA DE ÁRBOL Y MVC

- **Ejemplo**: se crea un componente Button con un texto. Se crea un nodo principal tipo VBox al que se le añade el botón. Se crea nueva Scene al que se le añade el VBox. Finalmente se configura el stage y se muestra.

```
public class ButtonExample extends Application {

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

    @Override
    public void start(Stage primaryStage) {

        Button button = new Button();
        button.setText(string: "Púlsame!");

        VBox vbox = new VBox(nodes: button);
        Scene scene = new Scene(parent: vbox);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

}
```

¿Falta algo para que esto funcione?

1 FUNDAMENTOS Y COMPONENTES

LLAMADA A EVENTOS

- POE: **P**rogramación **O**rientada a **E**ventos, es la base de las GUI
- Los componentes se tendrán que crear con **new()** y habrá que configurarles un *EventHandler* (veremos que esto NO es necesario cuando utilicemos FXML).
- En su definición se indica el tipo de evento que se maneja(ej: *ActionEvent*) y se sobrescribe el método handle(TipoEvento)
- Tenemos 2 alternativas (la 2ª con expresión lambda):

```
Button button = new Button(string: "Haz clic");
// Crea el EventHandler
EventHandler<ActionEvent> eventHandler = new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println(x: "¡Haz hecho clic en el botón!");
    }
};
// Agrega el EventHandler al botón
button.setOnAction(eh: eventHandler);
```

```
Button button = new Button(string: "Púlsame!");
button.setOnAction((event) -> {
    System.out.println(x: "¡Haz hecho clic en el botón!");
});
```

USAMOS ESTA!

1 FUNDAMENTOS Y COMPONENTES

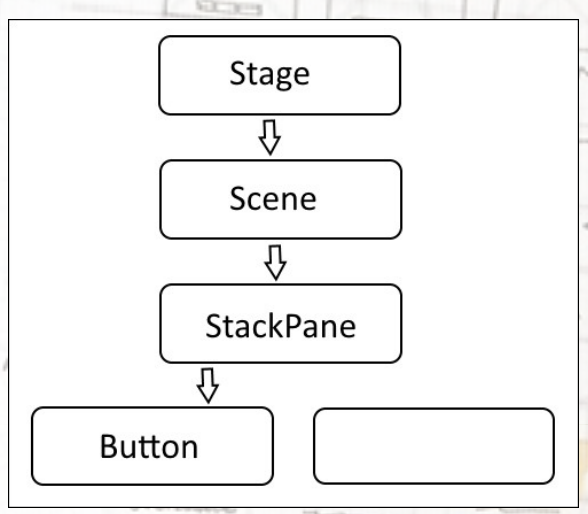
- **Ejemplo completo**: se añade un evento al pulsar el botón que saca un Alert

```
public class ButtonEjemplo extends Application {

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

    @Override
    public void start(Stage primaryStage) {


        Button buttonDesact = new Button();
        buttonDesact.setText(string:"Púlsame!");
        buttonDesact.setDisable(bln:true);


        Button button = new Button();
        button.setText(string:"Púlsame!");

        //Se usa .setOnAction y la notación Lambda
        button.setOnAction((event) -> {
            System.out.println(x: "Botón pulsado!");
            Alert alert = new Alert(at: Alert.AlertType.INFORMATION);
            alert.setTitle(string:"Info");
            alert.setHeaderText(string:"Cabecera");
            alert.setContentText(string:"Botón pulsado. Click en Aceptar para cerrar");
            // Mostrar la alerta y esperar a que el usuario la cierre
            alert.showAndWait();
        });

        VBox vbox = new VBox(nodes: buttonDesact,nodes: button);
        Scene scene = new Scene(parent:vbox);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

1 FUNDAMENTOS Y COMPONENTES

EVENTOS

Los eventos se “propagan” desde el nodo superior (Stage) hacia abajo (Scene, StackPane y el resto de nodos).
Esto permite que todos los nodos puedan capturar el evento y hacer algo al respecto.
Ej: que un mismo evento de teclado tenga distintos funcionamientos según el nodo utilizando un EventFilter (lo veremos luego)



El evento se monta sobre la función .setOn<nombreevento>

```
Objeto.setOn<nombreevento>(new EventHandler<tipoevento>){
...
}
```

Ejemplo:

```
Boton.setOnAction(new EventHandler<ActionEvent>){
..
}
```

1 FUNDAMENTOS Y COMPONENTES

Eventos de teclado KeyEvent: Se genera cuando se presiona o se suelta una tecla.

- KeyPressed: Se genera cuando se presiona una tecla.
- KeyReleased: Se genera cuando se suelta una tecla.
- KeyTyped: Se genera cuando se ingresa un carácter desde el teclado.

El evento se monta sobre la función .setOn<nombreevento>

```
Objeto.setOn<nombreevento>(new EventHandler<tipoevento>){
...
}
```

No hace falta poner el tipoevento si se usa notación lambda

```
Scene.setOnKeyPressed(event ->{
..
}
```

1 FUNDAMENTOS Y COMPONENTES

Eventos de ratón MouseEvent: Se genera cuando se interactúa con el ratón.

- MouseClicked: Se genera cuando se hace clic con el botón principal del ratón.
- MousePressed: Se genera cuando se presiona un botón del ratón.
- MouseReleased: Se genera cuando se suelta un botón del ratón.
- MouseEntered: Se genera cuando el ratón entra en el área de un nodo.
- MouseExited: Se genera cuando el ratón sale del área de un nodo.
- MouseMoved: Se genera cuando el ratón se mueve dentro del área de un nodo.
- MouseDragged: Se genera cuando el ratón se arrastra dentro del área de un nodo mientras se mantiene presionado un botón.

1 FUNDAMENTOS Y COMPONENTES

Eventos de acción ActionEvent: Se genera cuando ocurre una acción, como hacer clic en un botón.

- Action: Este evento se genera cuando se realiza una acción específica.

Eventos de cambio ChangeEvent: Se genera cuando cambia el valor de un nodo como por ejemplo un checkbox.

- Change: Este evento se genera cuando se realiza una acción específica.

- **Eventos de Foco** FocusEvent**:**
  - FocusGained: Se activa cuando un nodo obtiene el foco.
  - FocusLost: Se activa cuando un nodo pierde el foco.
- **Eventos de Arrastre y Soltar** DragEvent**:**
  - DragEntered: Se activa cuando un nodo se convierte en el objetivo de un evento de arrastre y soltar.
  - DragExited: Se activa cuando el evento de arrastre y soltar sale de un nodo.
  - DragOver: Se activa cuando se arrastra sobre un nodo.
  - DragDropped: Se activa cuando se suelta un objeto en un nodo.

1 FUNDAMENTOS Y COMPONENTES

Eventos de ventana WindowEvent: Se genera cuando se produce un cambio en el estado de una ventana, como abrir, cerrar o cambiar el tamaño.

- CloseRequest: Se activa cuando se intenta cerrar la ventana.
- Hidding y Hidden: Se activa cuando la ventana se está ocultando y después de haberse ocultado.
- Showing y Shown: Se activa cuando la ventana se está mostrando y después de haberse mostrado.

1 FUNDAMENTOS Y COMPONENTES

Colecciones: **FXCollections**: La clase FXCollections proporciona métodos para crear colecciones observables, como listas, mapas que controlan parejas clave->valor (ObservableMap) y conjuntos de datos para comprobar pertenencia(ObservableSet).

- **ObservableList**: Es la más utilizada.. ObservableList es una interfaz que extiende la interfaz java.util.List y agrega capacidades de observación (cambios). Se puede utilizar para almacenar y observar listas de elementos. Cuando los elementos se agregan, eliminan o modifican en la lista, la interfaz de usuario se actualiza automáticamente para reflejar esos cambios. Ejemplos:
  - Se puede utilizar una lista observable para rellenar los datos de un combobox y posteriormente si se actualiza la lista no habrá que "recargar" el combobox
  - Para guardar un conjunto de nodos (cajas, botones) para aplicarles algún cambio en conjunto
  - Leer los nodos hijos de un popup menú y acceder a ellos
  - Etc etc

1 FUNDAMENTOS Y COMPONENTES

- Vamos a ver los componentes y atributos más utilizados mediante **EJEMPLOS completos**
- Tener los apuntes mediante ejemplos cumplen 2 funciones, 1º No generar apuntes demasiado extensos y 2º Es la forma más fácil de entender un componente y su funcionamiento.
- Además, los ejemplos son auténticos BANCOS DE PRUEBAS, para ser modificados, vistos con el depurador, etc….
- Es necesario ver ejemplos de **creación con código** (aunque luego veremos cómo se crean mediante GUI)

- **POR FAVOR, SE RECOMIENDA IMPRIMIR EL CÓDIGO DE LOS EJEMPLOS PRINCIPALES PARA SER ESTUDIADO ;)**
- **SON EJEMPLOS MUY REPRESENTATIVOS DE TODO LO QUE SE PUEDE HACER CON JAVA FX**

2 EJEMPLOS

PROYECTOS DE EJEMPLO - Javafx-ejemplos

✓ Proyecto con un ***cargador por reflexión*** para probar cualquier ejemplo JavaFX: banco de pruebas para usar fuera de NetBeans (porque desde dentro siempre se puede lanzar por separado cada ejemplo con Shift+F6).

✓ Contiene mini-ejemplos de todos los componentes. Desde el principio vamos a mezclar conceptos porque es complicado exponer un ejemplo que no toque varios conceptos!! (hilos, eventos, layouts, etc..)

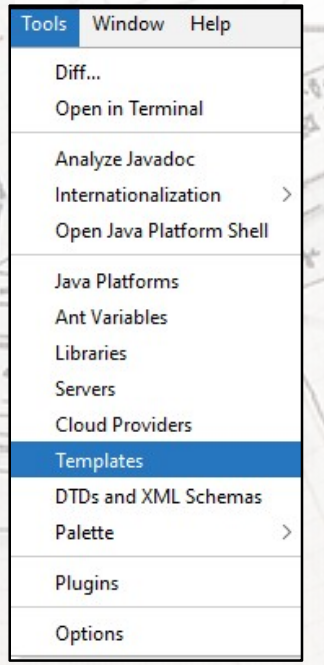
✓ **Muy importante!** Aunque la idea es que hagamos las GUI con un editor gráfico, todos estos ejemplos están creados y controlados directamente por código, pero nos sirven para conocer su funcionamiento y sus métodos principales.

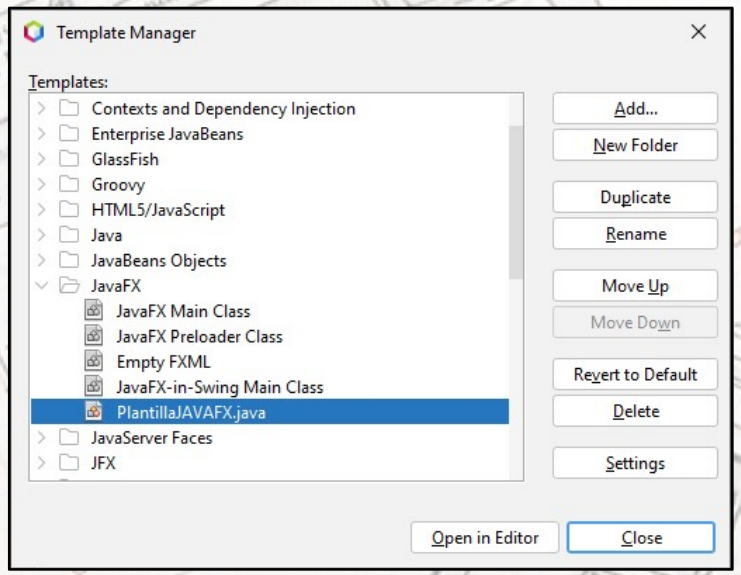
✓ Es muy sencillo añadir nuevos ejemplos (en el proyecto hay una plantilla).

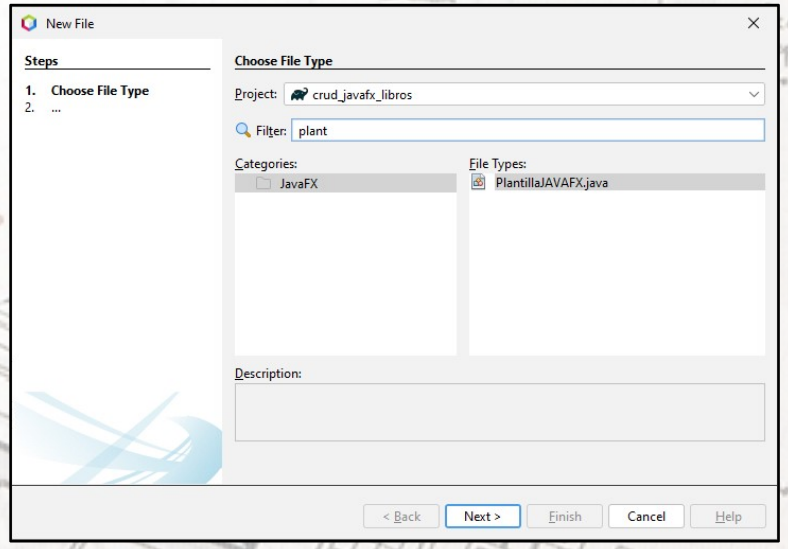
2 EJEMPLOS

PROYECTOS DE EJEMPLO - Javafx-ejemplos

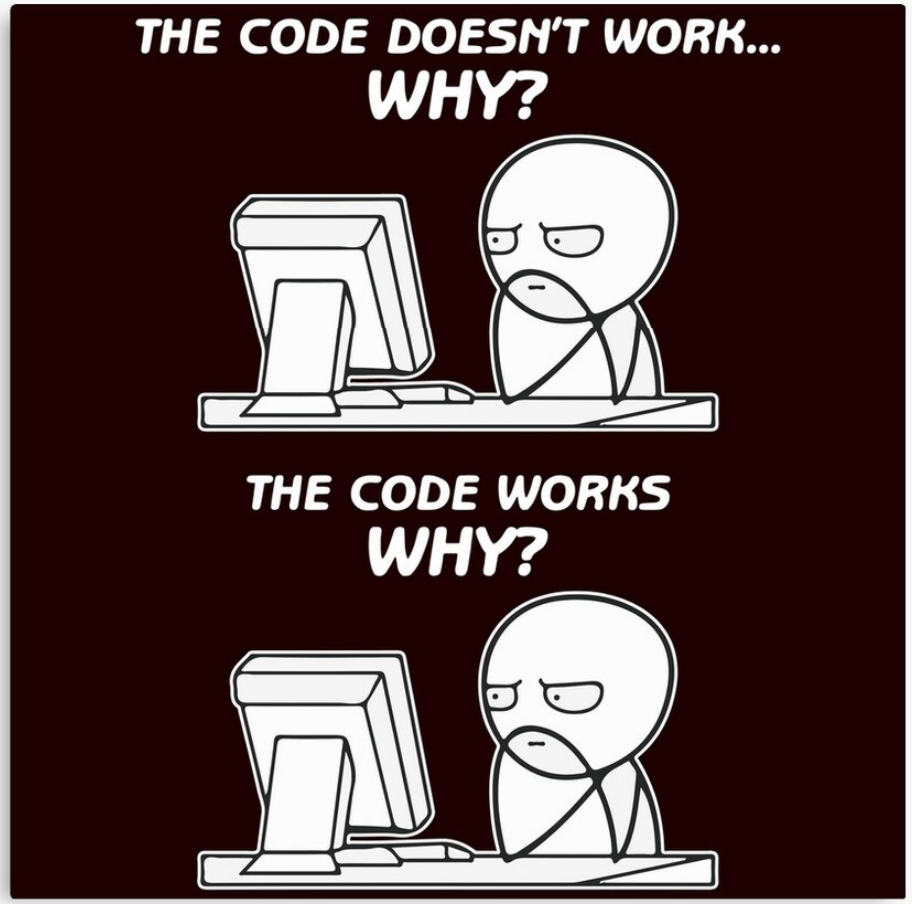
✓ Es muy sencillo añadir nuevos ejemplos (plantilla dentro del proyecto). Se puede crear una plantilla en NetBeans (Tools→Template, pinchamos en ADD y seleccionamos la clase Plantilla.java. A partir de ahora podremos crear un nuevo fichero desde File→New File (Ctrl+N)







2 EJEMPLOS



2 EJEMPLOS

AYUDA

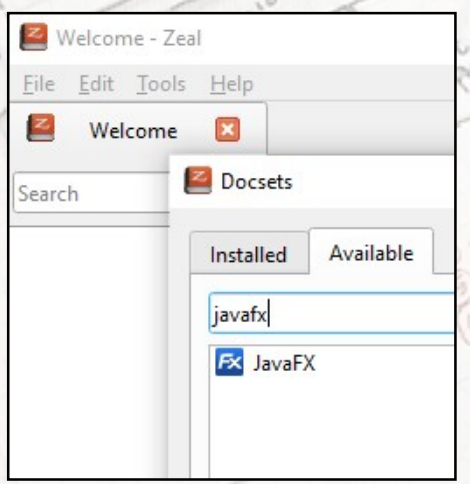
✓ OFICIAL: https://openjfx.io/ (nos vamos a apartado documentación)

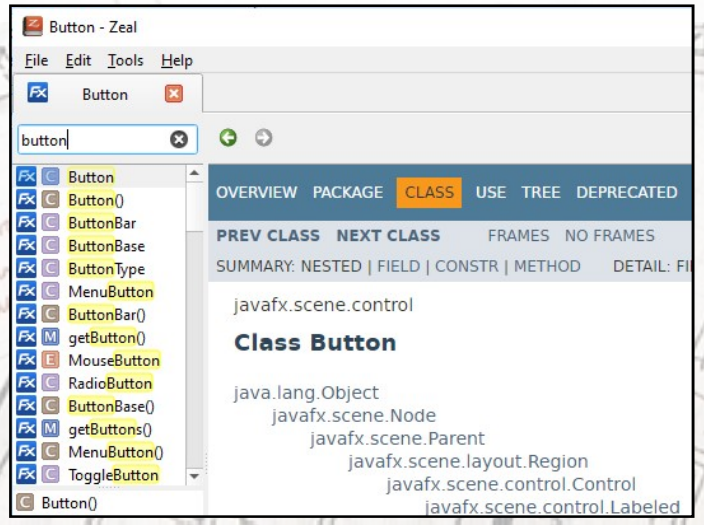
– https://openjfx.io/javadoc/23/ (OJO! Aparecen los módulos por categoría, las mismas categorías que se usan en build.gradle).

✓ ZEAL https://zealdocs.org/download.html (versión actual solo librerías JAVA, no JAVAFX. Podemos usarlo para PHP)

– Si no funciona, instalar librerías Visual C++ (https://aka.ms/vs/17/release/vc_redist.x64.exe)

– Añadir documentación: nos vamos a Tools→Docsets→Avaiable. Buscamos JavaFX y luego pinchamos en Download





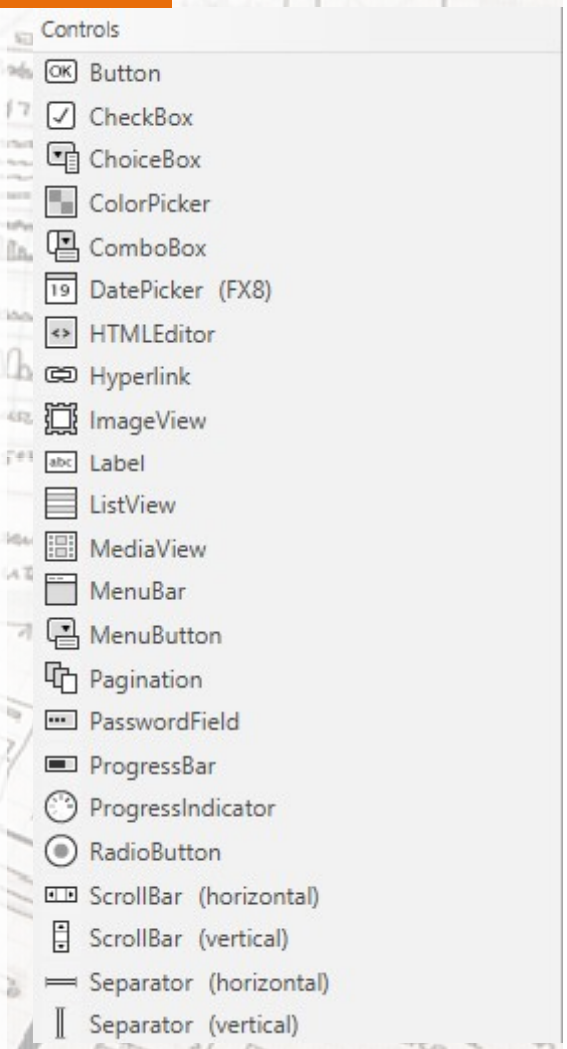
2 EJEMPLOS

CONTROLES BÁSICOS

✓ **Generales:** Se utilizarán de forma transversal en muchos ejemplos

– **Eventos** [com.javafx.eventos] : eventos de cierre de ventana y de teclado

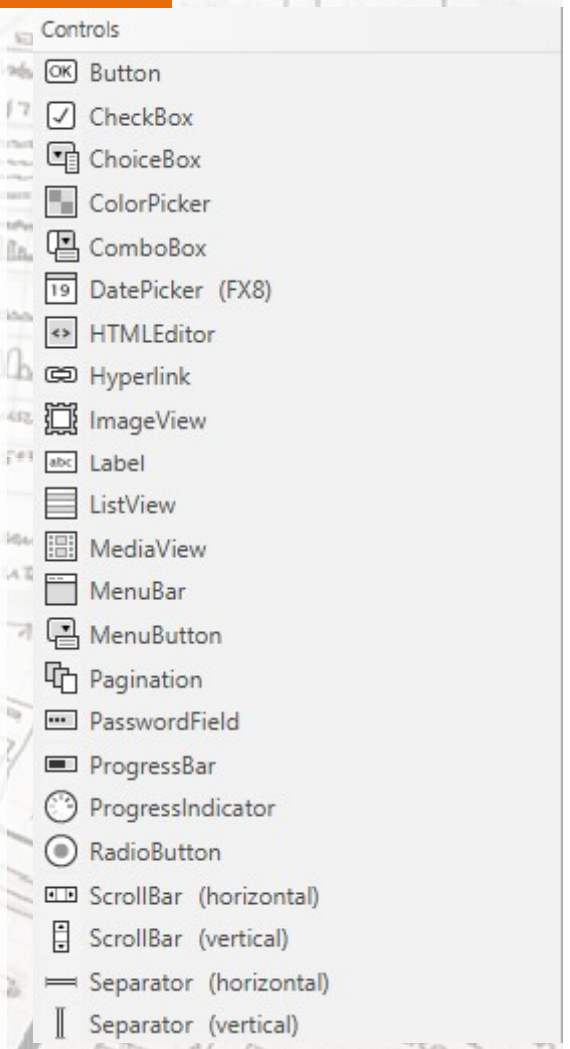
– **ObservableList** [com.javafx.observablelist]: contenedor lista observable manejada por un ChangeListener.



2 EJEMPLOS

CONTROLES BÁSICOS

- ✓ **Básicos [com.javafx.componentesbasicos]**:
  - **Button**: botón de acción. Otro: **[com.javafx.button]**
  - **Checkbox**: botón de activo/inactivo
  - **Combobox/Choicebox**: despliegue para selección. Otro: **[com.javafx.choiceBox] [com.javafx.comboBox]**
  - **DatePicker**: selección de fecha
  - **ImageView**: imágenes
  - **Label**: etiqueta
  - **PasswordField**: campo de password
  - **RadioButton**: botón de selección dependiente



2 EJEMPLOS

CONTROLES BÁSICOS

✓ **Básicos**:

– **TextField**: campo de texto para insertar datos **[com.javafx.componentesbasicos]**

Ya se pueden hacer los 6 primeros ejercicios (R1)

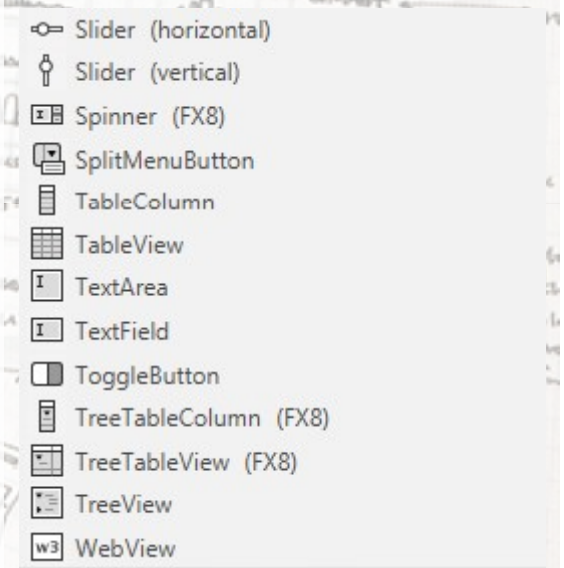
– **ListView:** permite añadir y eliminar una lista de elementos de forma sencilla **[com.javafx.listview]**(ver Anexo para personalizar)

– **TableView**: muy utilizado para mostrar datos **[com.javafx.tableview]** (ver Anexo para personalizar)

– **TextArea**: área de texto para insertar texto **[com.javafx.componentesbasicos]**

– **Text:** campo similar a label pero más configurable **[com.javafx.text]**

– **ToggleButton:** botón On/Off **[com.javafx.componentesbasicos]**



2 EJEMPLOS

MENÚS Y MISCELÁNEA

✓ Menús **[com.javafx.menubar]**:

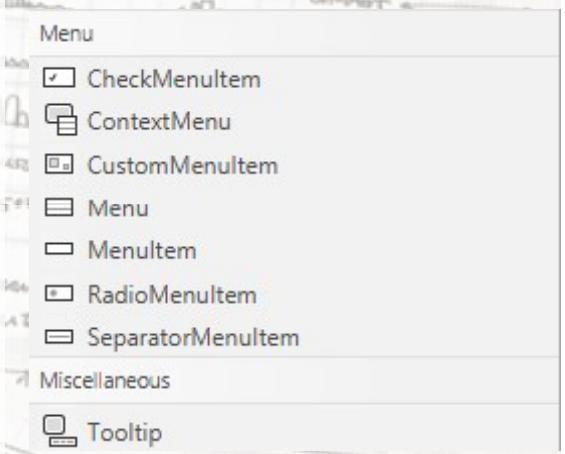
– **Menubar y MenuItem**: barra de menú superior

– **MenuItem**: componente de menú tanto Menubar como de ContextMenu **[com.javafx.contextmenu]**

– **MenuButton**: botón insertado dentro de un menú **[com.javafx.menubutton]**

– **Check/Radio/Separator-MenuItem**: checkbox, radiobutton y separador pero dentro de un menú.

Ya se pueden hacer el resto de ejercicios (R1)



2 EJEMPLOS

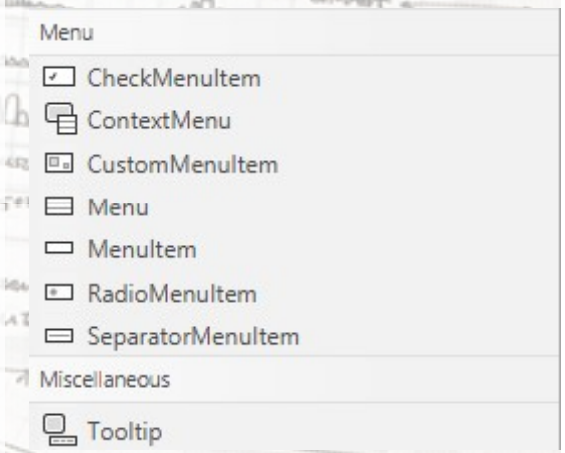
CONTROLES AUXILIARES

✓ Se pueden usar de forma transversal en todos los proyectos

– **Alertas** [com.javafx.alertas]: 4 tipos de alertas

– **Icono App**: [com.javafx.icono] : establecer icono en la barra de App.

– **FileChooser y Tooltip**: selector de ficheros [com.javafx.filechooser]



2 EJEMPLOS

CONTENEDORES PRIMARIOS

– **Stage** [com.javafx.stage] : hilo para cierre programado, pantalla completa, múltiples stages (modales y normales), decoración de barra superior, eventos de teclado.

– **Scene** [com.javafx.scene] : ejemplos de cursores con título, ancho y alto personalizado

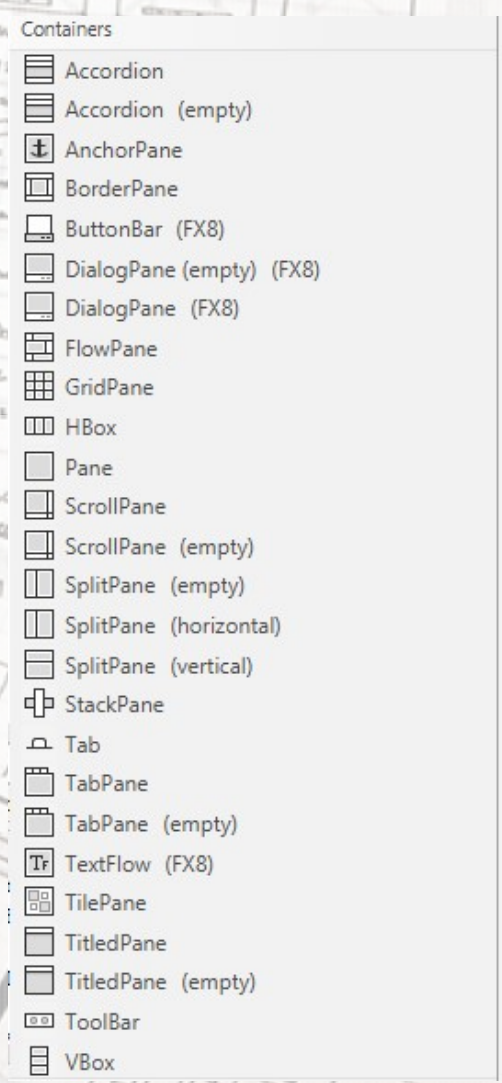
2 EJEMPLOS

CONTENEDORES SECUNDARIOS

– **VBox** y **Hbox** [com.javafx.contenedores]: permiten crear elementos dispuestos vertical u horizontalmente.

– **GridPane** y **BorderPane** [com.javafx.contenedores] y [com.javafx.gridpane] : el primero dispone todo como en una rejilla 2D, el segundo divide la ventana en cinco regiones: arriba, abajo, izquierda, derecha y centro (ideal para ciertas APP).

– **Pane** y **AnchorPane** [com.javafx.contenedores]: son paneles de disposición libre, aunque el segundo permite "anclar" elementos al borde del Pane y que los elementos sean responsive (sección Layout→Anchor Pane Constraint).



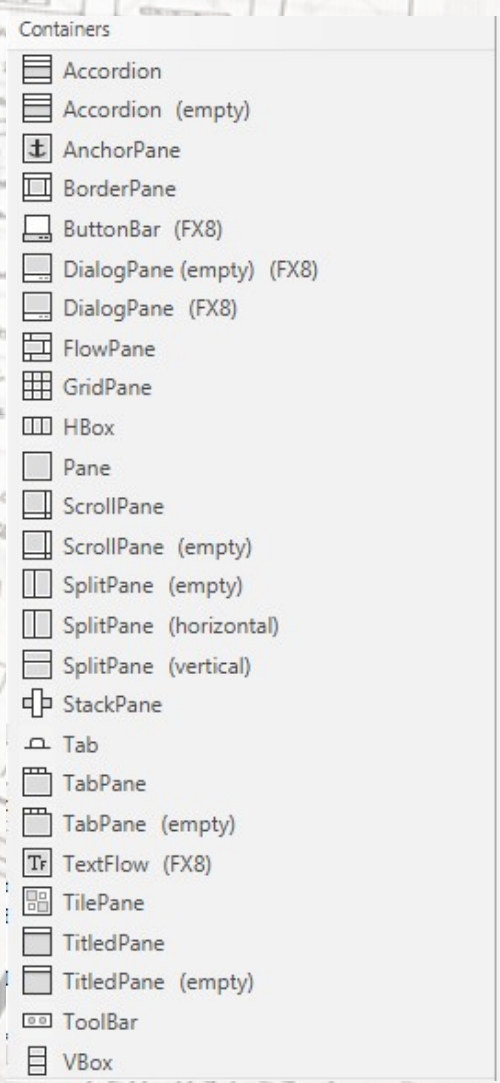
2 EJEMPLOS

CONTENEDORES SECUNDARIOS

– **Tabpane** **[com.javafx.tabpane]** : organización en pestañas

– **TitledPane** **[com.javafx.titledpane]** : despliega un panel al pinchar en su título

– **Toolbar** **[com.javafx.toolbar]** : barra de herramientas que normalmente agrupa botones

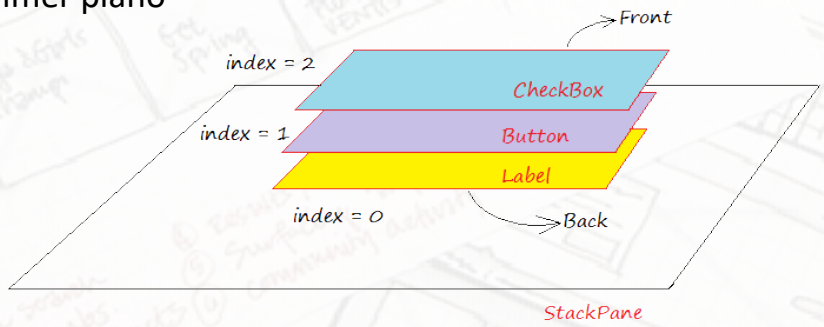


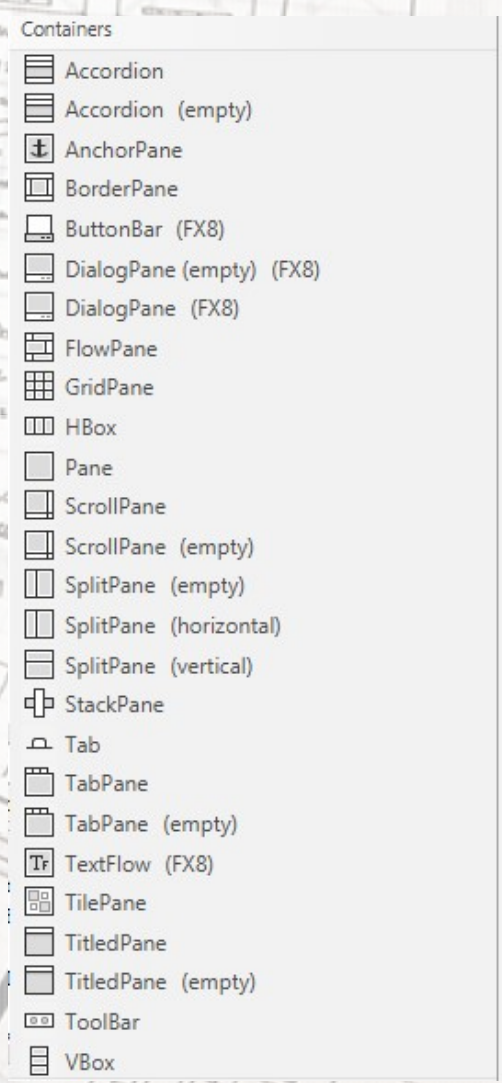
2 EJEMPLOS

CONTENEDORES SECUNDARIOS

– **Stackpane** [com.javafx.stackpane]: permite establecer elementos por "capas" (es el equivalente al JLayeredPane de Swing).

- Solo el que está en Front puede tener el foco, el resto serán visibles a no ser que se oculten con .setVisible()
- nodo.toBack(): envía atrás del todo
- nodo.toFront(): envía al primer plano

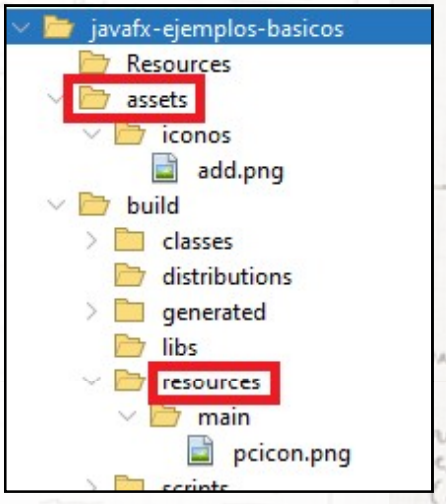




Ya se pueden hacer los ejercicios de la R2

3 ANEXO

AÑADIR RECURSOS - IMÁGENES



✓ Hay varias formas (lo vemos en **[com.javafx.cargaimagenes]**:):

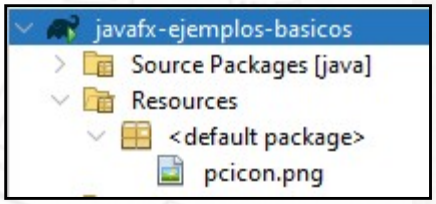
– Imagen desde URL o fichero tras pasarlo a URI (ojo, invierte barras en Windows)

Image imagen=new Image("http:.....", true);

Image imagen=new Image("file:C:/Users/...../loquesea.jpg", true);

– Imagen desde Resources (para recursos que se empaquetan dentro del jar y se acceden con ClassLoader):

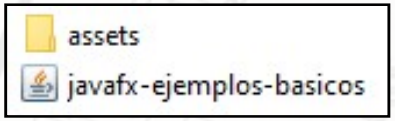
Image icon=new Image(getClass().getClassLoader().getResourceAsStream("pcicon.png"));



– desde Assets (recursos que NO van empaquetados, están FUERA del jar). FileInputStream se usa para cualquier tipo de fichero:

FileInputStream input=new FileInputStream("assets/iconos/add.png");

Image image=new Image(input);



Por tanto, cuando generemos el jar, junto a él tendrá que existir una subcarpeta llamada assets/iconos/ con add.png dentro para que pueda encontrar el fichero.

4 ANEXO

PERSONALIZAR LV Y TV

- ✓ **.setCellValueFactory()**: dice **QUÉ** datos usa del modelo, podrán ser directos:

– nameColumn.**setCellValueFactory**(new PropertyValueFactory<>("name"));

- ✓ Y También pueden ser personalizados (devolviendo otro Nodo)

```
- c5.setCellValueFactory(cellData -> {
        Button button = new Button("Púlsame!");
        button.setOnAction(event -> {
            RobotFoto r = cellData.getValue();//Acceso al objeto Modelo
            System.out.println("Has pinchado en: " + r.getNombre());
        });
        return new SimpleObjectProperty<>(button);
    });
```

4 ANEXO

PERSONALIZAR LV Y TV

✓ **.setCellFactory()**: dice **CÓMO** se personalizan los datos. Se puede utilizar para personalizar tanto un LV como un TV (aunque este último necesita cambiar ligeramente el código)

```
lvSeries.setCellFactory(e -> new ListCell<String>() {

        @Override

        //Se ejecuta cada vez que muestra un Item.

        protected void updateItem(String item, boolean empty) {

           super.updateItem(item, empty);

          .....//Aquí se personalizaría

        }

    });
```