PRÁCTICA 9. Programación Orientada a Eventos: JavaFX | ***2 H (1 Ses)***

Objetivos:

- Conocer el desarrollo de aplicaciones gráficas en Java (Graphical User Interfaces)
- Aprender a utilizar JavaFX.

Contenido:

El objetivo de esta práctica es iniciarse en el proceso de creación de aplicaciones con interfaces Gráfica así como experimentar los contenidos teóricos expuestos en el tema de teoría: Graphical User Interfaces (GUI).

Ejercicio 1. Ejemplo HelloWorld (Hecho)

El primer paso será crear un proyecto JavaFX (file → other → JavaFX project). Parámetros:

- Nombre del proyecto → Sesion 9.
- Seleccionamos las opciones con next (no con finish) y nombramos al paquete (es.unex.cum.mdp.sesion9) y Language (none).

Se creará un fichero Main.java donde existirán varios errores de ejecución → Faltan librerías de JFX. Configure → BuildPath → Libraries → Add Library → User Library → JavaFX (punto 2 del apartado anterior).

```
public class HelloWorld extends Application {
   public static void main(String[] args) {
            launch(args);
   }

   @Override
   public void start(Stage primaryStage) {
            primaryStage.setTitle("Hello World!");
            StackPane root = new StackPane();
            primaryStage.setScene(new Scene(root, 300, 250));
            primaryStage.show();
   }
}
```

Si ejecutamos, debería funcionar sin problema pues se ha configurado los parámetros de ejecución (paso 4 del apartado anterior).

- Comentamos la línea de CSS pues de momento no se va a utilizar. scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

Ejercicio 1. Botón y Eventos (Hecho)

A continuación vamos a crear un botón y vamos a capturar su evento Click. Cuando se pulse cambiaremos el título de la aplicación (propiedad setTitle de primaryStage)

```
Button btn = new Button();
btn.setText("Click me'");
btn.setOnAction(new EventHandler<ActionEvent>() {
  @Override
  public void handle(ActionEvent event) {
     ?????????
  }
});
root.setTop(btn);
```

Ejercicio 2. Etiquetas (Hecho)

A continuación crearemos una etiqueta que pondremos en la parte inferior del BorderPane y cuando pulsemos el botón cambiaremos su valor por la hora (propiedad setText("xxxxx") de la etiqueta).

Ejercicio 3. Cuadro de Texto. (Hecho)

A continuación añadir un cuadro de texto en la parte superior de la ventana y un botón en la inferior. De tal forma que cuando se pulse el botón se muestre si el número es par o impar.

```
btn.setOnAction(new EventHandler<ActionEvent>() {

	@Override
	public void handle(ActionEvent event) {
	//vamos a coger el valor de textfield y vamos a ver si es par o no
		try {
			int x= Integer.parseInt(t.getText());
			if (x%2==0)
				l.setText("Es par");
			else l.setText("Es impar");
		}catch (NumberFormatException e) {
			l.setText("Valor no valido");
		}

	}
});
```

Ejercicio 4. Diálogos

Se pueden introducir información por distintos medios (cuadro de texto, combobox, radiobutton, etc). Existe otro medio para introducir información: ventana emergente de diálogo.

Existen distintos tipos de diálogos: advertencia, error, confirmación, entrada de datos, selección de una opción, etc.

//Información
```
Alert alert = new Alert(AlertType.INFORMATION);
alert.setTitle("Diálogo de Información ");
alert.setHeaderText("Titulo del dialgo");
alert.setContentText("Buenos días!");
alert.showAndWait();
```

//Advertencia
```
Alert alert = new Alert(AlertType.WARNING);
alert.setTitle("Dialogo de Advertencia");
alert.setHeaderText("Título de diálogo");
alert.setContentText("Cuidado, formatearás el ordenador!");
alert.showAndWait();
```

```
//ERROR
Alert alert = new Alert(AlertType.ERROR);
alert.setTitle("Diálogo de Error");
alert.setHeaderText("Título de diálogo.");
alert.setContentText("Ooops, ALgo fué mal!");
alert.showAndWait();
```

```
//Confirmación
Alert alert = new Alert(AlertType.CONFIRMATION);
alert.setTitle("Dialogo de Confirmación");
alert.setHeaderText("Título del diálogo");
alert.setContentText("Esta todo correcto?");
Optional<ButtonType> result = alert.showAndWait();
if (result.get() == ButtonType.OK){
      //Hacer algo
} else {
    // Hacer algo
}
```

```
//Entrada de datos
TextInputDialog dialog = new TextInputDialog("walter");
dialog .setTitle("Dialogo de Entrada de datos");
dialog .setHeaderText("Título del diálogo");
Optional<String> result = dialog.showAndWait();
if (result.isPresent()){
          System.out.println("Tú nombre es: " + result.get());
}
```

Modificar el ejemplo anterior para que cuando se pulse un botón, lea un número. De tal forma que si es par se mostrará un mensaje de error y si es impar un mensaje de información. Al final en la etiqueta se mostrará el valor leído.

```
btn.setOnAction(new EventHandler<ActionEvent>() {
      @Override
      public void handle(ActionEvent event) {
```

```
            // primaryStage.setTitle("Dije Hola Mundo!");
            // l.setText(new Date().toString());
            Alert alert = new Alert(AlertType.CONFIRMATION);
            TextInputDialog dialog = new TextInputDialog("0");
            dialog.setTitle("Introduce un valor");
            dialog.setContentText("Introduce un valor");
            // Traditional way to get the response value.
            Optional<String> result = dialog.showAndWait();
            if (result.isPresent()) {
                  try {
                        int valor = Integer.parseInt(result.get());
                        if (valor % 2 == 0) {
                              Alert alert2 = new Alert(AlertType.ERROR);
                              alert2.setTitle("Error");
                              alert2.setContentText("Es par!");
                              alert2.showAndWait();
                        } else {
                              Alert alert3 = new Alert(AlertType.INFORMATION);
                              alert3.setTitle("Information Dialog");
                              alert3.setContentText("Genial es impar!");
                              alert3.showAndWait();
                        }
                        l.setText(result.get());
                  } catch (java.lang.NumberFormatException e) {
                        l.setText("dato no valido");
                  }
            } else {
                  l.setText("Has pulsado cancelar");
            }
      }
});
```

Java FXMVC: Java + FXML + Controller → SceneBuilder

Habitualmente las interfaces gráficas no se implementan introduciendo todos los paneles, controles y eventos manualmente → Se usa un plugins/aplicación gráfica. En el caso de JavaFX usaremos SceneBuilder y los ficheros fxml. Haremos el mismo ejemplo anterior pero usando SceneBuilder.

Paso 1: Creamos el fichero fxml → New → File → JavaFX → New FXML Document (borderPane) → **ej5.fxml** (en él se encuentra todos los contenedores, controles, configuraciones del diseño así como el fichero controller que será el que actue frente a los eventos).

Paso 2: Crearemos un nuevo main para no interferir con el anterior. → new → file → JavaFX → Java FX main class y copiamos el siguiente código que se encarga de cargar el fichero fxml. **ej5.java** (en él solo estará el cargar el fichero fxml con toda la información del diseño)

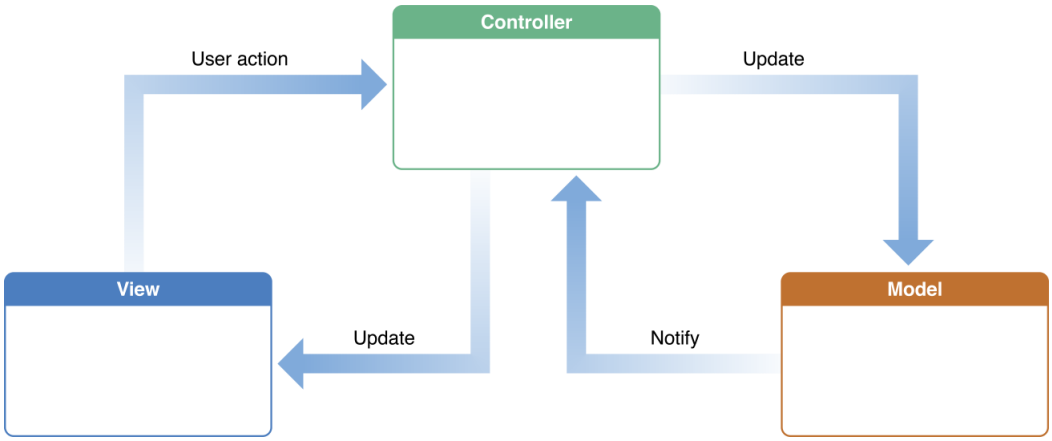
```
@Override
public void start(Stage primaryStage) {
  try {
      BorderPane root = (BorderPane) FXMLLoader.load(getClass().getResource
                  ("ej5.fxml")); //Ponemos el fichero fxml creado
      Scene scene = new Scene(root, 400, 400);
      //scene.getStylesheets().add(getClass().
      //getResource("application.css").toExternalForm());
      primaryStage.setScene(scene);
      primaryStage.show();
  } catch (Exception e) {
  }
}
```

Paso 3: Crearemos un fichero controller → new → file → class → **ej5_controller.java** (en él se capturarán todos los eventos y se establecerá las acciones oportunas.

- Incluimos en la cabecera de la clase **implements Initializable** e incluimos el método del interfaces sin hacer nada.

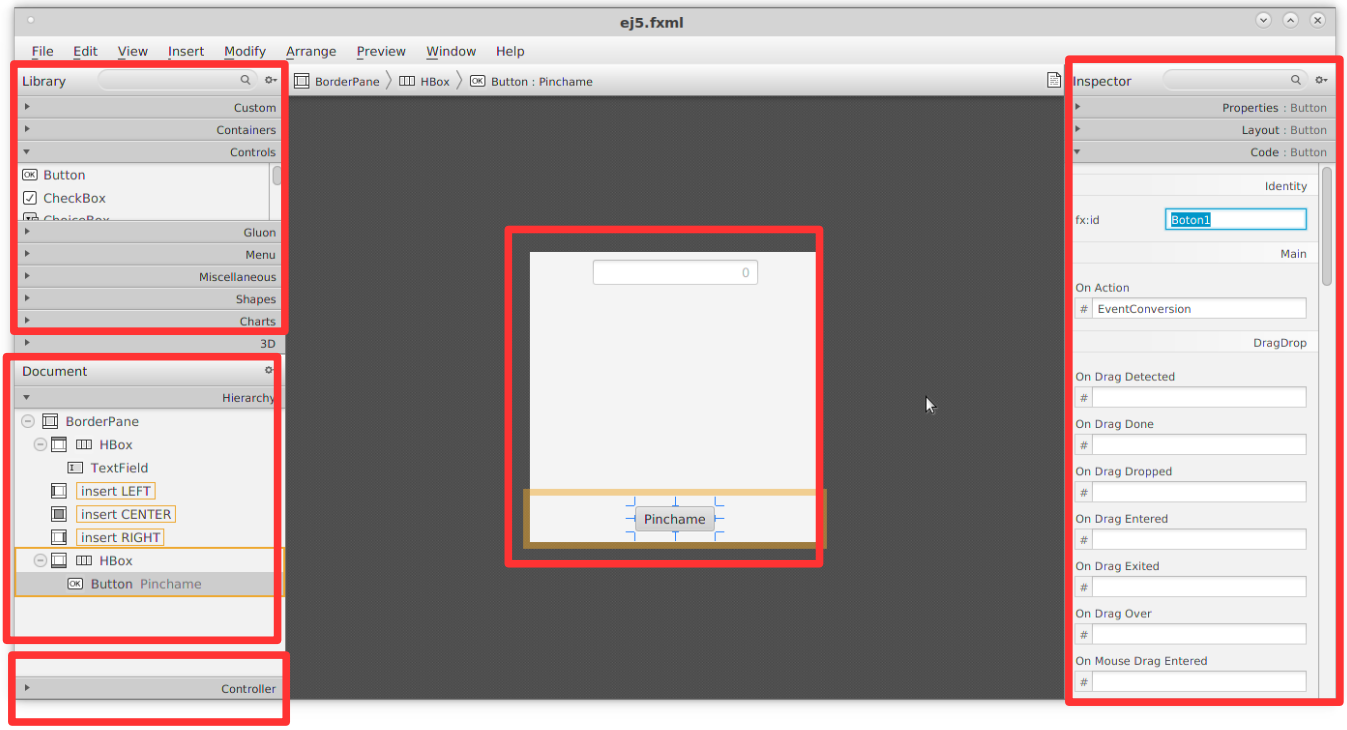
De este modo estableceremos el patrón MVC (modelo-vista-controlador) en JavaFX.

- Controller: Fichero java donde se capturan los eventos de la vista y actualiza la misma.
- View: Fichero .fxml con todos los contenedores y controles.
- Modelo: Lógica de nuestro programa.



Paso 4: Botón derecho sobre fxml → "Open With SceneBuilder" → Modificamos el fichero fxml con SceneBuilder. Esta formado como se puede ver en la figura de abajo:

- Library: Con todos los contenedores y controles (arrastrar y soltar)
- Document: Se muestra la estructura árborea con los contenedores y controles que hay en la aplicación.
- Controller: Indica el fichero controller de la aplicación.
- Properties: Muestra información de las propiedades del contenedor o controlador seleccionado.
  - Properties: Propiedades
  - Layout: Propiedades del diseño
  - Code: Propiedades del código



Procederemos a realizar los cambios en SceneBuilder:

- Seleccionamos el conteneder BorderPane (document hierarchy). A continuación en Properties y establecemos los valoresdea PrefWidth (300) y PrefHeight(300)
- Arrastramos un conteneddor Hbox al Top del BorderPane y cambiamos el alto del mismo usando el diseño mostrado en la parte central.
  - Modificamos en Properties la alineación a Center
- Arrastramos un control cuadro de texto (textField) al contender Hbox
  - En propiedades → Prompt Text → 0
  - En propiedades → Alineamiento → Baseline_Rigth
  - En Layout → fx:id → "CuadroTexto"
- Arrastramos un contenedor Hbox al Botton del BordePane y cambiamos el alto del mismo usando el diseño mostrado en la parte central.

    - Modificamos en Properties la alineación a Center
- Arrastramos un botón al Hbox inferior
    - En propiedades → modificar el valor Text (Pinchame)
    - En Code → fx:id → Boton1
    - En Code → onAction→ EventConversion
- En el apartado Controller
    - Indicamos el nombre del fichero controlador → **es.unex.cum.mdp.sesion9.ej5_controller**
- Salvamos

Problema: Todos los ficheros se han actualizado menos el fichero controlador.

- En SceneBuilder seleccionamos → View → Show Sample Controller → y copiamos el texto y lo pegamos en Eclipse.

Una vez que ya esta todo, procedemos a darle la lógica a la aplicación en el método EventConversion

```
@FXML
void EventConversion(ActionEvent event) {
	// Model Data
	try {
		String valor = CuadroTexto.getText();
		float value = Float.parseFloat(valor);
		value = value * 166.386F;

		// Show in VIEW
		CuadroTexto.setText(String.valueOf(value));
	} catch (java.lang.NumberFormatException e) {
		CuadroTexto.setText("Error no valido");
	}
}
```

Ejercicio 6.

Incluir un nuevo botón en la parte inferior de tal forma que cuando se pulse pide una ventana de diálogo un nombre y lo ponga en el cuadro de texto.

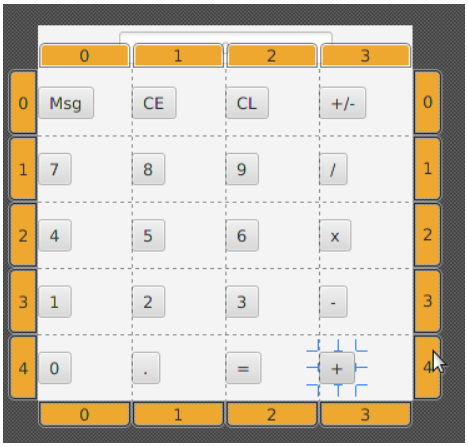
Ejercicio 7. Calculadora

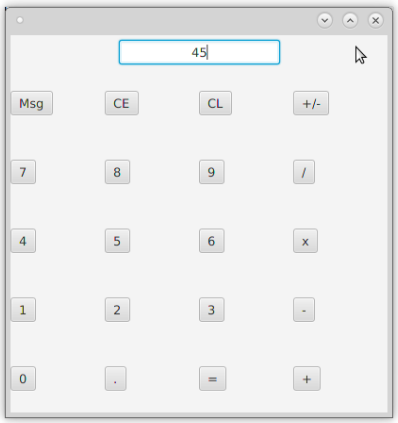
Creación del esqueleto de la aplicación.

- Crear un nuevo fichero **fxml** (ej7Calculadora.fxml con borderPane)
- Crear un nuevo fichero **ej7Calculadora**.java donde se tendrá el modelo de la calculadora (lógica)
- Crear un nuevo fichero **ej7Controller**. Java donde se realizará el control de los eventos (debemos incluir **implements Initializable**)
- Crear un nuevo fichero main JavaFX class **ej7Main**.java donde se encargará de lanzar la aplicación (copiar el del proyecto 5 y cambiar los datos oportunos).
- Crear un fichero denominado **ej7Estilos**.css donde se pondrá las reglas css.

Configuración de la interfaz en SceneBuilder.

- Sobre el BorderPane definir el tamaño de 320 Width y 200 Height
- En el top incluir un AnchorPane de 320 de Width
  - Incluir un textField con 320 de Width
  - Con text con valor 0
  - **Con valor fx:id → text1**
- En el Center incluir un GridPane formado por 5 filas y 4 columnas (botón derecho sobre GridPane → AddRow o AddColumn)
- En cada Celda del GridPane incluir un botón
  - Cambiar la propiedad Text indicando el valor tal y como aparece en la figura
  - Cambiar la propiedad en Code de **fx:id** con su valor (boton1, botonMas, botonMenos, …)
- Finalmente seleccionamos View → Show Sample Controller Skeleton y copiamos en el fichero Controller de Eclipse





Si ejecutamos a esta altura el proyecto deberíamos mostrarnos la imagen de la izquierda sin ninguna operabilidad. Vamos a continuar mejorando el diseño y luego abordaremos la lógica de la misma.

Paso 1: Incluimos el siguiente código de CSS donde definimos tres estilos: uno para todos los botones, otro denominado especial y otro denominado operador.

```
Button {
    -fx-background-color:
        #000000,
        linear-gradient(#7ebcea, #2f4b8f),
        linear-gradient(#426ab7, #263e75),
        linear-gradient(#395cab, #223768);
    -fx-background-insets: 0,1,2,3;
    -fx-background-radius: 3,2,2,2;
    -fx-padding: 14 37 14 37;
    -fx-text-fill: white;
    -fx-font-size: 12px;
    -fx-font-family: monospace;
}
.especiales {
    -fx-background-color: #000000, linear-gradient(#ff5400, #be1d00);
}
.operador {
    -fx-background-color: #000000, linear-gradient(#52A839, #548246);
```

```
}
```

Con esto todos los botones deberán verse de color azul.

Paso 2: Cambiaremos de color los botones operadores (operador) y las teclas especiales (especiales)

- Seleccionamos los tres botones especiales → Propiedades Botón → En StyleClass seleccionamos **Especial**
- Seleccionamos los botones operadores→ Propiedades Botón → En StyleClass seleccionamos **Operador**

Paso 3: El cuadro de texto debe modificarse con las teclas pero vamos a proceder a capturan los eventos de los botones numéricos

- Seleccionamos, por ejemplo el botón con el número 1 → code → On Action → pondremos un nombre para ese evento → **BotonNumerico**
- Nos habrá generado en el fichero fxml para el botón 1 onaction=”BotonNumerico”
- Ahora queda trasladarlo al Controller → View Controller Skeleton y pegamos únicamente el código onAction

```
@FXML
void BotonNumerico(ActionEvent event) {


}
```

- Nos queda por hacer la lógica. La idea es añadir el número 1 al cuadro de texto

```
            text1.appendText("1");
```

IMPORTANTE: Se debería realizar lo mismo para el resto de botones numerico pero no tiene como mucho sentido, tener que copiar el mismo código con valores numéricos distintos → reutilizaremos la función botonNumerico y dependiendo del botón pulsado nos quedamos con su número. Dos formas: en base al valor textual o en base al nombre del botón

```
	void BotonNumerico(ActionEvent event) {
		// En base al valor del text
		String text = ((Button) event.getSource()).getText();
		if (text.equals("1")) {
			text1.appendText("1");
		} else if (text.equals("2")) {
			text1.appendText("2");
		}

		// En base al nombre del boton
		if (event.getSource() instanceof Button) {
			Button button = (Button) event.getSource();
			if (button.equals(btn1)) {
				text1.appendText("1");
			} else if (button.equals(btn2)) {
				text1.appendText("2");
			}
		}

	}
```

¿Alguna mejora se os ocurre a este código?

Llegado a este punto ya queda por realizar la lógica separando cada una de las partes del patrón MVC.

- Se deberá realizar la implementación de una calculadora con cuatro métodos: add, sub, mult y div que realizará las operaciones oportunas con float.
- Cuando se pulse +, -, *, /: se deberá mantener el número (porque el texto debe borrarse) para realizar la operación
- Cuando se pulse = se realizará la operación usando el modelo (calculadora)
- Msg: Mostrará un mensaje de bienvenida en un cuadro de diálogos
- CE: Borrar los números introducidos (tanto si es el primer operador como el segundo)
- CL: Borrar los números del operador actual.

Se deberá entregar la calculadora realizando las operaciones. Se tendrá en cuenta la funcionalidad de la misma.