DESARROLLO DE INTERFACES
2º DAM

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

TEMA 2-5 – CSS JAVAFX

2-5 CSS JAVAFX

Vamos a utilizar la potencia de las CSS amplicar nuestra APP JavaFX: nos permiten **personalizar**, cumplir reglas de **usabilidad** por sí solas (coherencia) y ayudar en los avisos de **validación de campos**.

Como todo en JavaFX, la dificultad está conocer la enorme cantidad de variables/atributos que podemos aplicar.

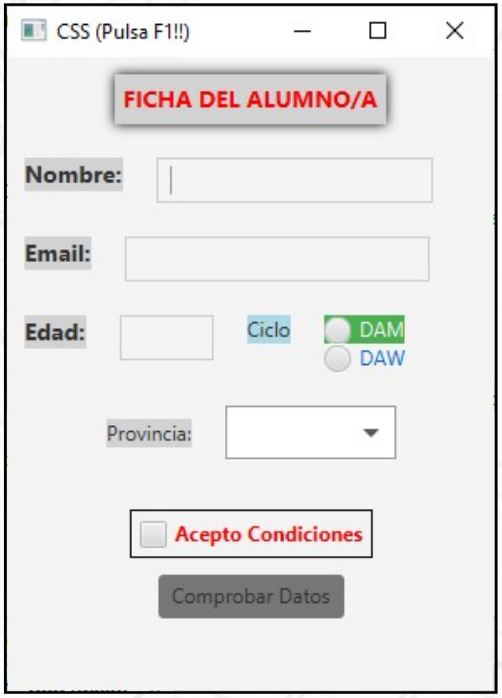
Vemos toda la teoría y en la diapositiva 10 empezamos a practicar.

- ✓ 1 CSS
- ✓ 2 EJEMPLOS

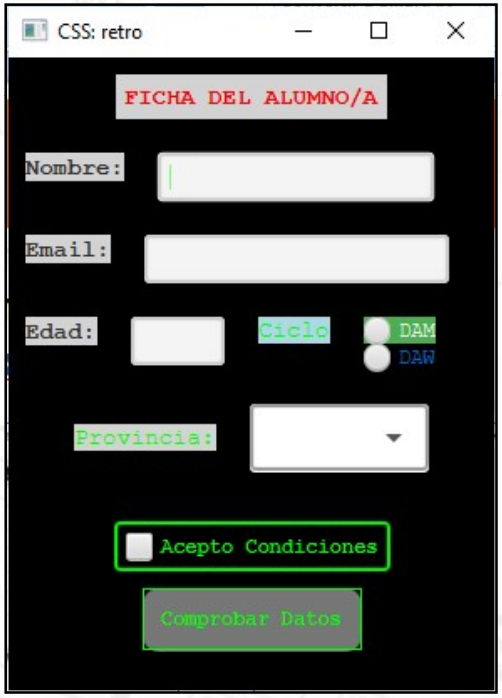
2- EJEMPLOS

✓ Vemos ejemplos de CSS y de validación de campos+CSS sobre el proyecto [**javafx-ejemplos-CSS**]

Estilos aplicados desde SB



Asignación desde JAVA



Cambio de estilos dinámico

1 - CSS

QUÉ ES CSS?

✓ **CSS**: Las ***C**ascade **S**tyle **S**heet* (Hojas de Estilo en Cascada) son formatos de estilo que se aplican por propagación (cascada) de un padre hacia sus hijos.

✓ Versión actual es CSS3 (no solo se aplica en web)

✓ Etiquetas con formato para JAVAFX: **-fx-**<estilo>

✓ Documentación oficial CSS para FXML (se puede lanzar desde SB con F4):

– https://openjfx.io/javadoc/22/javafx.graphics/javafx/scene/doc-files/cssref.html (vemos propiedades por Nodo, tratando de ver las más comunes)

✓ Una cosa curiosa es que por CSS también sirve para modificar cualquier parámetro de SB, uno muy visual son los EFFECTS:

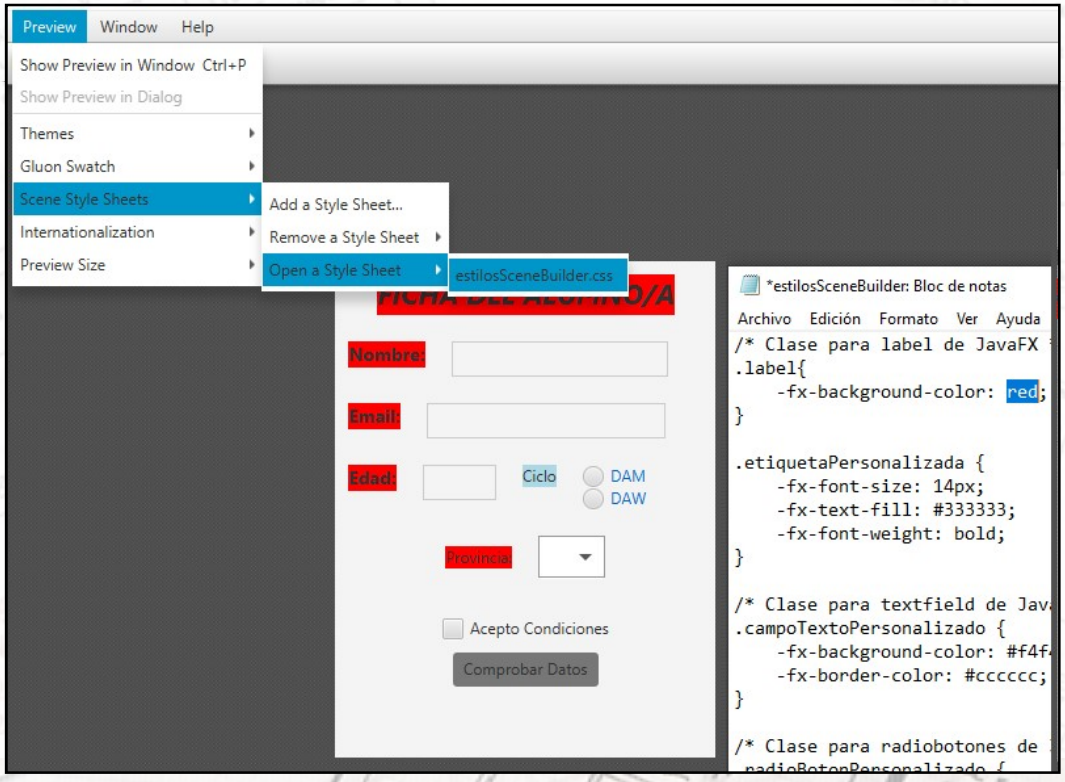
– https://openjfx.io/javadoc/22/javafx.graphics/javafx/scene/effect/package-summary.html

1 - CSS

✓ Permite dejar gran parte del diseño para el final y así ahorrar tiempo y código: tomamos base sin diseño y vamos aplicando CSS hasta que nos guste...

1 - CSS

– Aunque SB tiene una forma de abrir las CSS (Preview→Scene Style Sheets→Open a Style Sheet), lo más sencillo es abrirlo con **cualquier editor** (Notepad++,VSCode,IntelliJ,Netbeans,etc.. **guardar y observar cambios de forma automática incluso en Preview [Ctrl+P]**)



1 - CSS

✓ CLASES, SUBCLASES, IDENTIFICADORES Y PSEUDOCLASES

– Las **CLASES** de estilos **AGRUPAN ESTILOS** para aplicarlos en bloque a un **mismo** tipo de nodo.

– Por ejemplo, podemos usarlo para aplicar un estilo a TODOS los **label** en bloque de mi formulario.

– Se deben indicar el nombre de la **clase**, en minúsculas, precedido por un punto (.): **.label**{….}

– Las **SUBCLASES** de estilos especializan aún más las CLASES. Por ejemplo: aplicamos un estilo a los label principales de un formulario y otro al resto, o incluso podemos dar estilos a distintos nodos pero que comparten propiedad. Ej: creamos una subclase .error y la podemos aplicar de forma condicional a muchos nodos vía JAVA.

1 - CSS

✓ **CLASES, SUBCLASES, IDENTIFICADORES Y PSEUDOCLASES**

– Los **IDENTIFICADORES** me permiten poner excepciones a las **CLASES ,** aplicando un estilo distinto. Ej: le damos un estilo especial a la etiqueta de Email. Tiene precedencia sobre clases, subclases y pseudoclases.

– Las **PSEUDOCLASES** permiten diferenciar estilo según estado. Por ejemplo, para cambiar el estilo del button cuando se posiciona el ratón sobre él usamos hover:

✓ :hover: Se aplica cuando el ratón está sobre el botón.

✓ :pressed: Se aplica cuando el botón está siendo presionado.

✓ :focused: Se aplica cuando el botón tiene el foco.

✓ :disabled: Se aplica cuando el botón está deshabilitado.

1 - CSS

✓ **Ejemplo:** Asignamos CSS desde SB y añadimos estilos directamente desde JAVA (para el proyecto se puede hacer de cualquier forma).

1 - CSS


```css
/* Clase para NODOS de JavaFX */
.label{
    -fx-background-color: lightgreen;
}
.button {
    -fx-background-color: #FF7777;
    -fx-font-size: 14px;
}
.check-box {
    -fx-font-size: 12px;
    -fx-padding: 5;
    -fx-font-weight: bold;
    -fx-border-color: black;
}
/*-------------------------------------------------------*/
/* SubClases */
.label-especial{
    -fx-background-color: darkgray;
    -fx-font-size: 14px;
    -fx-text-fill: #333333;
    -fx-font-weight: bold;
}
.textfield-especial {
    -fx-background-color: #f0f0f0;
    -fx-border-color: #CCCC00;
    -fx-effect: dropshadow(gaussian, rgba(0,0,0,0.75), 10, 0, 0, 2);
}
/*-------------------------------------------------------*/
/* Identificador para combobox de JavaFX */
#label-personalizada {
    -fx-background-color: #ffffff;
    -fx-border-color: #999999;
}
/*-------------------------------------------------------*/
/* Pseudoclases*/
.button:hover {
    -fx-background-color: lightblue;
}
.button:pressed {
    -fx-background-color: #0066cc; /*Azul oscuro*/
}
```

1 - CSS

CÓMO APLICARLAS

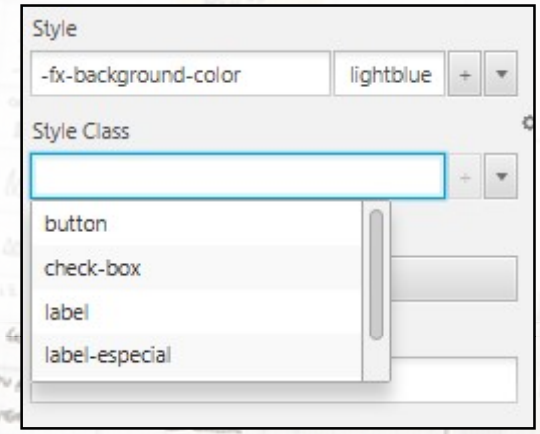
✓ Lo más sencillo es mediante SceneBuilder.

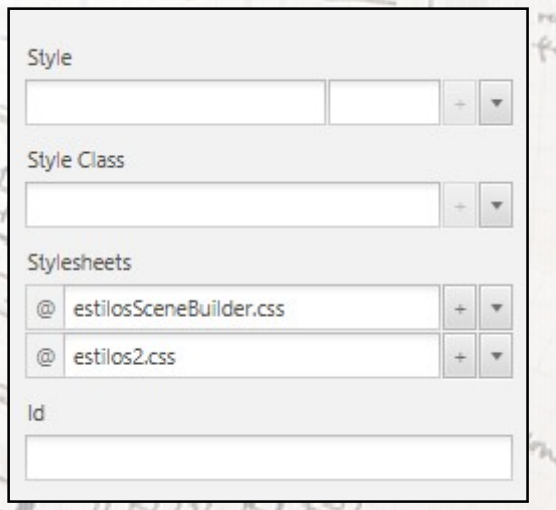
- **Style**: se aplica el estilo directamente con su valor. Ej: ***-fx-background-color*** con valor ***red.*** Podemos añadir todos los que queramos.

- **Style Class**: Tiene sentido usar este parámetro cuando usamos una subclase, ya que por defecto un nodo cogerá el estilo que le corresponde según la CSS. Si usamos SubClase sí hay que poner nombre. Ej: ***.label-especial***.

- **Stylesheets**: se puede añadir uno o varios ficheros css externos. Se asigna al NODO a partir del cual sus hijos tomarán la CSS. Suele asignarse al nodo padre (root).

- **Id**: se asigna un identificador (que tendrá que estar definido en la CSS). Ej: ***#label-personalizada***





1 - CSS

- ✓ **Desde SB:**

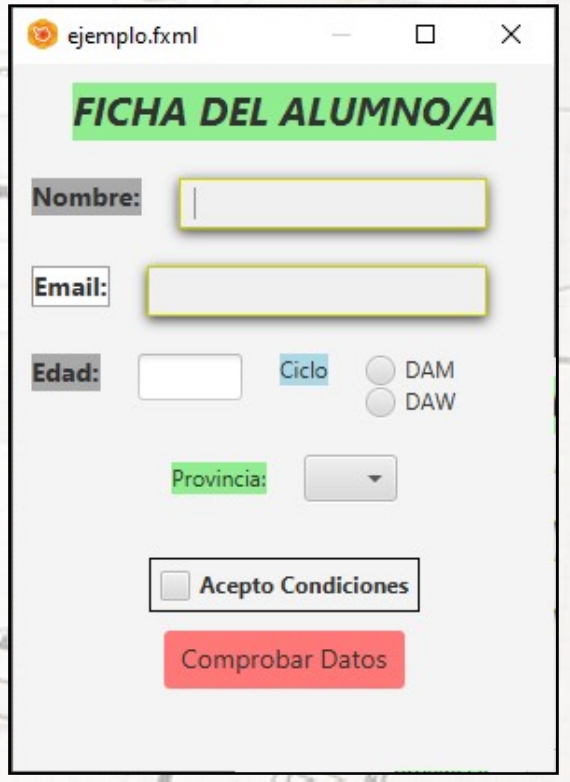
  - Estilos directos: etiqueta “Ciclo”

  - Clases (automáticamente vía asignación de css): **label** (lightgreen), **button** y **checkbox**

  - Subclases : etiquetas Nombre y Edad (**label-especial** en gris/negro). Textfield especial para nombre e email (**textfield-especial**)

  - ID: Email usa además un identificador **(label-personalizada en** blanco/negro/borde**),** el cuál tiene más precedencia que la clase y subclase.

  - Subclases: **button:hover** y **button:pressed**



1 - CSS

✓ Desde JAVA (javafx-ejemplos-CSS.css)

– Se aplica estilos directo a damRadioButton por código

– Además se añade CSS que incluye estilos por id (**#titulo-personalizado**) y por clases (**radio-button y check-especial**)

```
/* ID para label de JavaFX */
#titulo-personalizado {
    -fx-font-size: 14px;
    -fx-font-weight: bold;
    -fx-text-fill: blue;
    -fx-padding: 5;
    -fx-effect: dropshadow(three-pass-box, #00FF00, 10, 0, 0, 0);
    /* Efecto de sombra: tipo de blur(three-pass-box), color, radio, x, y, spread */
}
/* Clase para radio*/
.radio-button{
    -fx-background-color: blue;
}
/* SUBClase para radio*/
.check-especial{
    -fx-font-style: italic;
}
```

1 - CSS

CÓMO APLICARLAS DESDE JAVA

✓ Se aplica por código (añadiendo o borrando estilos):

- scene.**getStylesheets().add**(getClass().getResource("estilosNetBeans.css").toString()); (necesita un String)
- scene.getStylesheets().remove(0);//Elimina la primera css añadida desde aquí
- scene.getStylesheets().clear();//Las limpia todas

✓ Para aplicar un estilo directo se usa **.setStyle()**

✓ Para aplicar un Id (y por tanto utilizar la asignación por id) se utiliza el método nodo.setId y para añadir clase se utiliza nodo.getStyleClass().add:

- Cajatexto**.setId**(“titulo-personalizado”);
- condicCheckBox**.getStyleClass().add**(“check-especial");
- condicCheckBox**.getStyleClass().clear()** //Limpia estilo asignado

1 - CSS

CÓMO APLICARLAS – VALIDACIÓN DE CAMPOS

✓ El método **.setStyle** permite asignar estilos directos separados por ";"
Se suele utilizar para **Validación de Campos** como método complementario. Ej: borde de 2px rojo si error, sin estilo si todo OK:

```
//Comprobación campos vacíos TextField
for (TextField campo : camposTexto) {
    String texto = campo.getText();
    if (texto.isEmpty()) {
        System.out.println("El campo está vacío: " + campo.getId());
        campo.setStyle(string: "-fx-border-color: red; -fx-border-width: 2px;");
        campo.requestFocus();
        return false;
    } else {
        campo.setStyle(string: "");
        System.out.println("El campo contiene texto: " + campo.getId() + " - " + texto);
    }
}
```

1 - CSS

CÓMO APLICARLAS – VALIDACIÓN DE CAMPOS (javafx-ejemplos-CSS.v2.css)

✓ Hay un método mucho más elegante y es utilizando los validadores mediante la consulta de un Property (validationResult):

```css
/*Estilo común de error*/
.error {
    -fx-background-color: #444444; /*fondo oscuro */
    -fx-text-fill: #ffffff; /* texto claro */
    -fx-border-color: #ff4d4d; /*Borde rojo brillante para indicar error */
    -fx-border-width: 2px;
}
```

```
//Validación CSS
//Aplicamos estilos CSS en función del resultado de la validación
vSnombre.validationResultProperty().addListener((observable, oldValue, newValue) -> {
    if (newValue.getErrors().isEmpty() && newValue.getWarnings().isEmpty()) {
        nombreTextField.getStyleClass().remove("error");
    } else {
        if (!nombreTextField.getStyleClass().contains("error")) {
            nombreTextField.getStyleClass().add("error");//evita volver a aplicar estilo
        }
    }
});
```

1 - CSS

CÓMO APLICARLAS – VALIDACIÓN DE CAMPOS

✓ Y ya puestos, por qué no recorrer todos los Validators y aplicar css a todos los controles (además aplicamos un effect):

```
//Requiere crear el efecto DropShadow
for (ValidationSupport vS : validadores) {
    vS.validationResultProperty().addListener((observable, oldValue, newValue) -> {
        Set<Control> controles = vS.getRegisteredControls();//Cogemos control/es
        System.out.println(controles.size());
        for (Control c : controles) {//Recorremos Set
            System.out.println(c);
            if (newValue.getErrors().isEmpty() && newValue.getWarnings().isEmpty()) {
                c.getStyleClass().remove("error");
                c.setEffect(creaDropShadow(Color.GREEN));
            } else {
                if (!c.getStyleClass().contains("error")) {
                    c.getStyleClass().add("error");//evita volver a aplicar estilo
                    c.setEffect(creaDropShadow(Color.RED));
                }
            }
        }
    });
}
```

1 - CSS

CÓMO APLICARLAS – VALIDACIÓN DE CAMPOS

✓ Con todo lo que ya sabemos, deberíamos saber hacer distintas validaciones CSS:

- Que sólo aplique los CSS al final (al pulsar el botón)
- Aplicar CSS solamente sobre el campo en el que estamos actualmente
- Aplicar CSS al campo anterior al cambiarnos de control (pérdida de Foco)
- Aplicar CSS por tipo de Validador o por tipo de Node
- Etc…