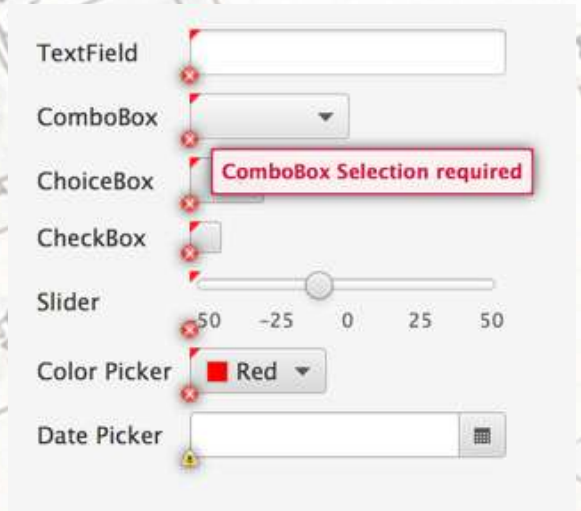
DESARROLLO DE INTERFACES
2º DAM

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



TEMA 2-4 – VALIDACIÓN DE CAMPOS

2-4 VALIDACIÓN

Vamos a ver distintas formas de validar campos, formas no gráficas (que ya conocemos) y formas más adornadas de presentar fallos en formularios.

git clone https://github.com/MolinaJM/DI-T2-5-ValidacCampos.git

- ✓1 VALIDACIÓN BÁSICA
- ✓2 VALIDACIÓN CONTROLFX: VALIDATORS
- ✓3 ICONOS PERSONALIZADOS
- ✓4 VALIDACIÓN CSS

1 – VALIDACIÓN BÁSICA

1) FORMA BÁSICA

✓ Posibles comprobaciones:

- **Cadenas vacías**: isEmpty o con equals.” “
- **Longitud 0 o longitud determinada**: función length
- **Valores nulos**: null. Ej: radio, combo, checkbox no seleccionado
- **Seleccionados** (checkbox/radio): isSelected()
- **Items concretos e**n combobox/choicebox/list/tableview mediante .getSelectionModel(): .getSelectedItem() o .getSelectedIndex()
- **Expresiones regulares**: Ej: CP, email, etc…
- **Rangos numéri**cos: Ej: edad entre 1 y 99, etc..
- **Tipos de validación mediante Excepciones**

1 – VALIDACIÓN BÁSICA

1) FORMA BÁSICA

✓ Uso de **expresiones regulares**: https://4geeks.com/es/lesson/expresiones-regulares-java

✓ En qué **momento** se comprueba?? Alternativas:

– Por bloques y a la vez: al pulsar un botón general de envío, se comprueba TODO.

– Por pasos: al pasar de un campo a otro se pueden comprobar de forma individual:

- Si se hace mediante ratón o teclado (se puede usar un listener y la propiedad de .focusedProperty para comprobar ganancia o pérdida de foco)

- Si se hace solo mediante teclado tendremos que ir pasando el foco de forma manual usando .requestFocus()

Lo vemos en el proyecto **[javafx-validacion-campos.v1]**

2 - VALIDACIÓN CONTROLFX

2) CONTROLSFX: VALIDATORS

✓ Utilizamos una librería externa basada en JavaFX: ControlsFX (https://mvnrepository.com/artifact/org.controlsfx/controlsfx)

– **implementation 'org.controlsfx:controlsfx:11.2.2'** (lo añadimos en build.gradle)

✓ Clase ***ValidationSupport*** permite validar controles de múltiples formas (se puede usar 1 solo para todo, pero para aprender empezamos usando uno por cada control)

✓ Ejemplo de uso: Queremos comprobar que un **edadTextfield** cumple condiciones numéricas (es número y cumple un rango):

– Creamos un manejador de validaciones (**ValidationSupport**), que permite crear distintos validadores (**.createValidator**), registrarlos/asociarlos (**.registerValidator**) y esas reglas generan resultados **.getValidationResult()** que permiten tomar decisiones y además pueden generar avisos de forma gráfica.

2 - VALIDACIÓN CONTROLFX

2) **VALIDATORS** ¿cómo registrar un validador?:

✓ public final <T> void **registerValidator**(Control **c**, Validator<T> **v**): **registra** el validador **v** en el campo **c** para validar un cierto tipo de dato.

– Control **c**: campo (cualquier nodo hijo de javafx.scene.control.Control) al que se aplica la validación. Ej: edadTextfield

– Validator<T> **v**: v es el validador (Validator) y es una interfaz parametrizada que define cómo se hará la validación. El tipo genérico T se refiere al tipo de dato que se va a validar en el campo. Por ejemplo, si estamos un campo de texto, T podría ser String.

✓ Se pueden CREAR validadores y registrarlos después o crearlos en el mismo registro.

✓ Se puede incluir un segundo parámetro que de estar a falso oculta el molesto **triangulito rojo**.

2 - VALIDACIÓN CONTROLFX

2) VALIDATORS

✓ ***ValidationResult*** es la clase que se usa para almacenar resultados de validación, que pueden incluir mensajes de error y detalles sobre los campos que no pasaron la validación. El método para **RECOGER** errores es **validationSupport.getValidationResult()**.

✓ Se puede consultar distintos resultados por cada control procesado. Se dividen en tipos o SEVERIDAD (Severity): errores, infos y warnings. Messages los recoge todos.

```
for (ValidationSupport validationSupport : validadores) {
    ValidationResult resultados = validationSupport.getValidationResult();
    System.out.println("Validador: " + validationSupport.getRegisteredControls());
    System.out.println("Errores: " + resultados.getErrors());
    System.out.println("Infos: " + resultados.getInfos());
    System.out.println("Mensajes: " + resultados.getMessages());
    System.out.println("Warnings: " + resultados.getWarnings());
}
```

2 - VALIDACIÓN CONTROLFX

2) VALIDATORS: ¿cómo se muestra?

Indicador de que tiene al menos 1 **validador asociado**

GraphicValidationDecoration : añade un .setGraphic al nodo etiqueta etiqNombre

Indicador de **Severidad** (ERROR, INFO, OK o WARNING)

GraphicValidationDecoration: Icono extra asociado a una etiqueta invisible (etiqIconoNombre).

2 - VALIDACIÓN CONTROLFX

2) TIPOS DE VALIDATORS

✓ **EmptyValidator**: usado para verificar si un campo de entrada está **vacío** (solo para campos de texto, no vale ni para checkbox, ni radios). Solo devuelve el Severity ERROR.

✓ **RegexValidator**: utiliza **expresiones regulares para validar el contenido** de un campo de entrada. Puede devolver cualquier tipo de Severity. Es útil para verificar formatos específicos, como direcciones de correo electrónico, números de teléfono, mails, códigos compuestos (DNI,..), etc.

✓ **PredicateValidator**: Predicado es una función que toma si incumple la condición dada me genera un tooltip gráfico (rellena automáticamente en array Severity.ERROR). Me permite definir una validación personalizada mediante una expresión lambda (λ). Puede ser usado para **validar cualquier tipo de condición específica**: rangos de números, combinaciones de condiciones, etc.

✓ **CustomValidator**: El CustomValidator permite crear **validaciones mucho más complejas** y específicas que no están cubiertas por los validadores estándar. Además puede devolver distintos resultados

Los 3 primeros se pueden registrar directamente, el último, no

2 - VALIDACIÓN CONTROLFX

2) VALIDATORS – 4 Ejemplos:

✓ Validator.create**Empty**Validator("Nombre vacío"): valida si está o no vacío.

✓ Validator.create**Regex**Validator("Solo se admiten números. Meter al menos 1 dígito!!", "^\\d+$", Severity.ERROR): valida una expresión regular, en el caso de NO CUMPLIRSE aparecerá un icono (según sea el enumerado Severity.<tipoerror>) y un tooltip con el texto indicado.

✓ Validator.create**Predicate**Validator(texto -> texto.length() >= 5 && texto.length() <= 10, "El nombre debe tener entre 5 y 10 caracteres", Severity.WARNING);. Los parámetros son (una/varias condiciones, texto tooltip gráfico, severidad).

λ value, es de tipo String

✓ Validator<String> **custom**Validator = (control, value) -> {
    if (value.length() !=5) {
        return **ValidationResult**.fromError(control, "CP son 5 caracteres!!");
    }
};
validationSupport.registerValidator(textFieldCP, customValidator);

2 - VALIDACIÓN CONTROLFX

2) VALIDATORS

✓ **ValidationResult** es la clase que se usa para almacenar resultados de validación, que pueden incluir mensajes de error y detalles sobre los campos que no pasaron la validación. Algunos de los métodos y propiedades comunes de la clase ValidationResult para **ENVIAR** errores según tipo (SEVERITY):

– ValidationResult.**fromError**(Control c, String message): Crea una instancia de ValidationResult con un mensaje de error especificado.

– ValidationResult.**fromInfo**(Control c, String message): Crea una instancia de ValidationResult con un mensaje de info especificado.

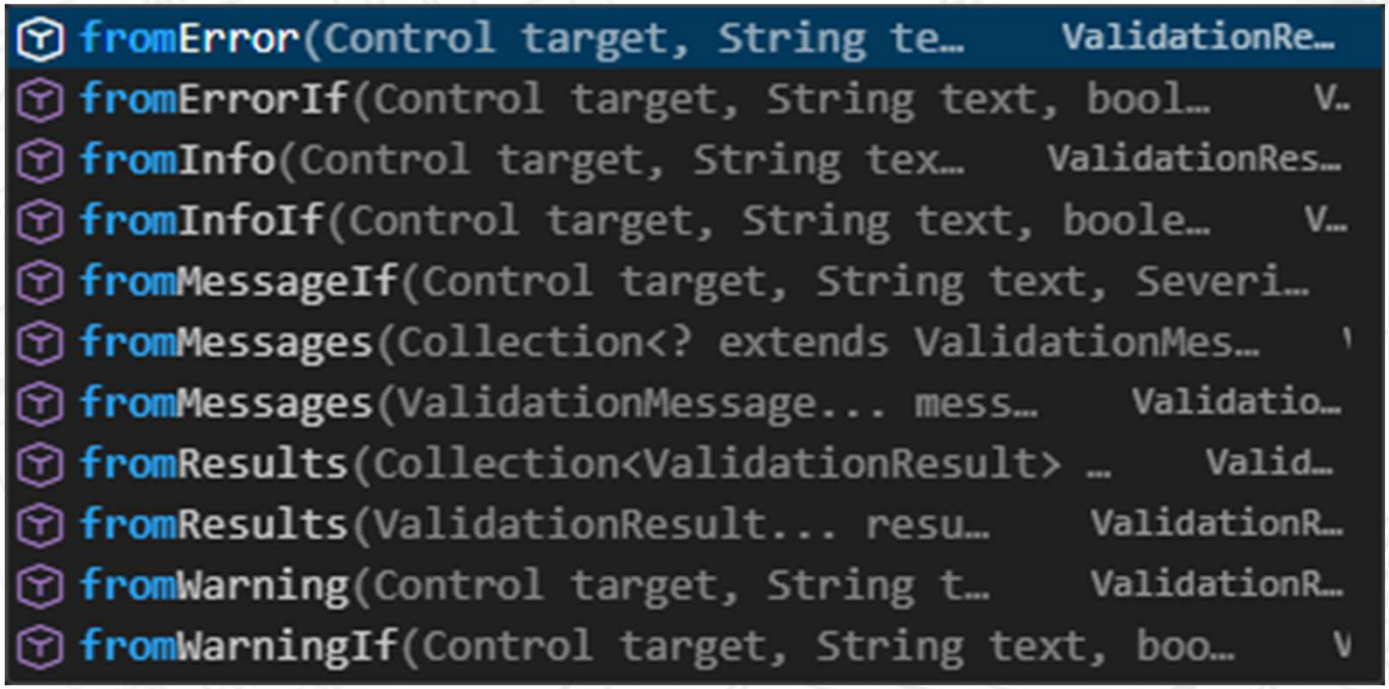
– ValidationResult.**fromWarning**(Control c, String message): Crea una instancia de ValidationResult con un mensaje de warning.

✓ El método ValidationResult.fromMessages registra todos los mensajes anteriores, es una especie de almacén común: NO USAREMOS ESTE MÉTODO.

2 - VALIDACIÓN CONTROLFX

2) VALIDATORS

✓ VARIANTES: Los 4 métodos anteriores tienen una versión If cuyo tercer parámetro puede ser la condición que se evalúa.



2 - VALIDACIÓN CONTROLFX

2) VALIDATORS

- ✓ CustomValidator y su alternativa con If:

```
Validator<String> customValidator = (control, value) -> {
        if (value.length() !=5) {
                return ValidationResult.fromError(control, “CP son 5 caracteres!!");
        }
};
```

OTRA FORMA DE ESCRIBIRLO

```
Validator<String> customValidator = (control, value) -> {
        return ValidationResult.fromErrorIf(control, “CP son 5
caracteres",value.length() != 5));

};
```

Se reescribe el código, es otra forma más compacta de representar lo mismo.

2 - VALIDACIÓN CONTROLFX

3) ICONOS PERSONALIZADOS

✓ La clase *GraphicValidationDecoration* comprueba el tipo de error y asigna un gráfico (método .setGraphic) a un nodo hijo de la clase Labeled: Label, Button, Checkbox, RadioButton y ToggleButton.

✓ No confundir el párrafo anterior con el hecho de que podemos "asociar" el GraphicValidationDecoration con cualquier ValidationSupport de cualquier control directamente sin haber error de por medio.

✓ Pasos: 1) Se crea un nuevo objeto *GraphicValidationDecoration* llamado decorador, 2) se sobrecarga el método *applyValidationDecoration* y dependiendo del error enviado se crea un nodo con una imagen u otro (error o acierto) y 3) Se asigna el decorador al validador:

– vSnombre.setValidationDecorator(decorador);

2 - VALIDACIÓN CONTROLFX

3) ICONOS PERSONALIZADOS

✓ Hay que tener en cuenta que cualquier tema relacionado con la carga de gráficos necesita ser lanzado en un **nuevo hilo** (este decorador es un caso y los adornos gráficos del Validator también es otro). Esto es debido al retardo en la carga de la interfaz y su configuración inicial.

✓ Es una forma muy versátil de personalizar errores con iconos o imágenes personalizadas. Lo vemos todo en el proyecto **[javafx-validacion-campos.v2]**

2 - VALIDACIÓN CONTROLFX

4) VALIDACIÓN + CSS

✓Las CSS se utilizan para añadir estilo a los componentes, pero además se pueden utilizar para ayudar a resaltar los errores en la validación con un simple if-else (próximo tema)