



Universidad
Nacional
de Quilmes

CONSTRUCCIÓN DE INTERFACES DE USUARIO

**TRANSFORMERS
FILTERS
ADAPTERS**

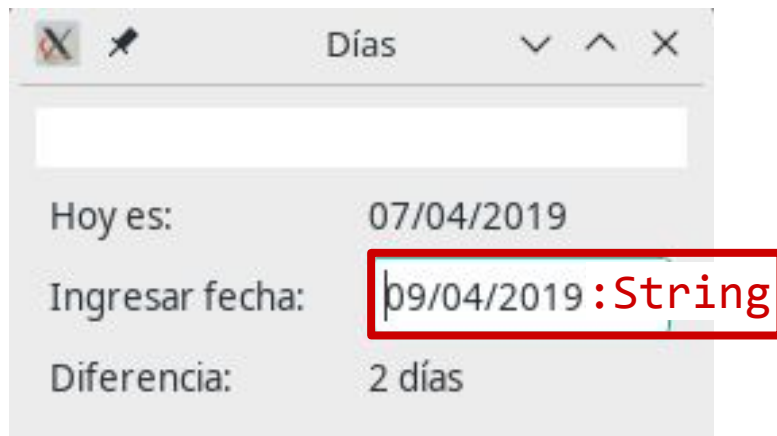


Problemas de *Binding*

Modelo

Vista

```
@Observable  
class TimeCounter {  
    var now =  
        LocalDate.now()  
    var date: LocalDate?  
    var diff: Int  
}
```



A screenshot of a Java Swing window titled "Días". The window contains a text input field at the top. Below it, there are three rows of text: "Hoy es:" followed by "07/04/2019", "Ingresar fecha:" followed by "09/04/2019:String", and "Diferencia:" followed by "2 días". The text "09/04/2019:String" is highlighted with a red box, indicating a binding error where the date string is not being converted to a LocalDate object.

Problemas de *Binding*

Al querer *bindear* el input de la vista con el atributo del modelo podemos llegar a tener problemas

- ▶ De *Model* a *View* a veces funciona por el `toString()`
- ▶ De *View* a *Model* explota por tipado

```
org.eclipse.core.databinding - 0 - Could not change value of  
ar.unq.edu.ui.arena.ej_contador_dias.TimeCounter@6850b758  
.another Java.lang.IllegalArgumentException: argument type mismatch
```

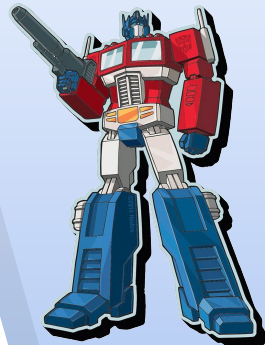
¿Cómo se resuelve?

Utilizando un objeto que transforme:

- ▶ El objeto de modelo en *String*
- ▶ El *String* en objeto de modelo

En Arena existe una herramienta para eso llamada *Transformer*

Transformers ⇒ Definición



```
class DateTransformer : ValueTransformer<LocalDate, String> {  
    private val pattern = "dd/MM/yyyy"  
    private val formatter = DateTimeFormatter.ofPattern(pattern)  
  
    override fun getModelType() = LocalDate::class.java  
    override fun getViewType() = String::class.java  
  
    override fun modelToView(valueFromModel: LocalDate): String =  
        valueFromModel.format(formatter)  
  
    override fun viewToModel(valueFromView: String): LocalDate {  
        try {  
            return LocalDate.parse(valueFromView, formatter)  
        } catch (e: DateTimeParseException) {  
            throw UserException("Fecha incorrecta, usar $pattern", e)  
        }  
    }  
}
```

Transformers ⇒ Uso

```
class TimeCounterWindow : MainWindow<TimeCounter> {  
    constructor(model: TimeCounter) : super(model)  
    override fun createContents(mainPanel: Panel) {  
        title = "Días"  
        mainPanel.asColumns(2)  
        Label(mainPanel) withText "Hoy es:"  
        Label(mainPanel)  
            .bindTo("today").setTransformer(DateTransformer())  
  
        Label(mainPanel) withText "Ingresar fecha:"  
        TextBox(mainPanel) with {  
            bindTo("date").setTransformer(DateTransformer())  
        }  
        Label(mainPanel) withText "Diferencia:"  
        Label(mainPanel) with { bindTo("diffStr") }  
    }  
}
```

Transformers ⇒ Resultado

✕ ✎ Días ▼ ▲ ✕

Hoy es: 07/04/2019

Ingresa fecha:

Diferencia: 2 días

✕ ✎ Días ▼ ▲ ✕

Fecha incorrecta, usar dd/MM/yyyy

Hoy es: 07/04/2019

Ingresa fecha:

Diferencia: 2 días

Problemas que sigue habiendo

- ▶ Es necesario verificar que el valor a transformar tengas las características deseadas
- ▶ Si no cumple, ¿qué debería pasar?
- ▶ ¿Podemos mostrar un error?
- ▶ ¿Podemos dar un valor por defecto?
- ▶ Quizás se le está planteando demasiada responsabilidad al Transformer

¿Cómo se resuelve?

Podemos minimizar los errores utilizando un objeto que **Filtre** la entrada, permitiendo ingresar exclusivamente los valores correctos (aunque no siempre lo resuelve del todo).

Filtros comunes:

- ▶ Sólo Números
- ▶ Sólo caracteres alfanuméricos
- ▶ Máximo X caracteres
- ▶ etc...

Filters ⇒ Definición

```
class DateTextFilter : TextFilter {  
    override fun accept(event: TextInputEvent): Boolean {  
        val expected =  
            "[0-9]{0,2}\\./?[0-9]{0,2}\\./?[0-9]{0,4}"  
            .toRegex()  
  
        return event.potentialTextResult.matches(expected)  
    }  
}
```

01/12/2018
01/12/201
01/12/20
01/12/2
01/12/
01/12
01/1
01/
01
1

Filters ⇒ con class

```
class TimeCounterWindow : MainWindow<TimeCounter> {  
    constructor(model: TimeCounter) : super(model)  
    override fun createContents(mainPanel: Panel) {  
        title = "Días"  
        mainPanel.asColumns(2)  
        Label(mainPanel) withText "Hoy es:"  
        Label(mainPanel)  
            .bindTo("today").setTransformer(DateTransformer())  
        Label(mainPanel) withText "Ingresar fecha:"  
        TextBox(mainPanel) with {  
            withFilter(DateTextFilter())  
            bindTo("date").setTransformer(DateTransformer())  
        }  
        Label(mainPanel) withText "Diferencia:"  
        Label(mainPanel) with { bindTo("diffStr") }  
    }  
}
```

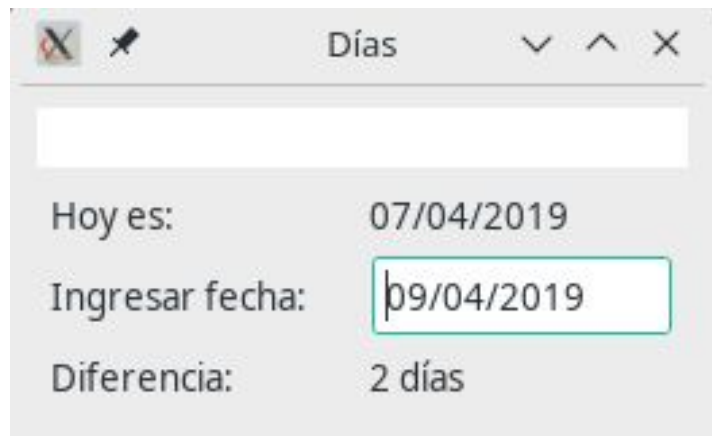
Filters ⇒ in block

```
class TimeCounterWindow : MainWindow<TimeCounter> {  
    override fun createContents(mainPanel: Panel) {  
        /* ... */  
        Label(mainPanel) withText "Ingresar fecha:"  
        TextBox(mainPanel) with {  
            withFilter {  
                val exp = "[0-9]{0,2}\\/?[0-9]{0,2}\\/?[0-9]{0,4}"  
                it.potentialTextResult.matches(exp.toRegex())  
            }  
            bindTo("date").setTransformer(DateTransformer())  
        }  
    }  
}
```

Filters ⇒ CustomBox

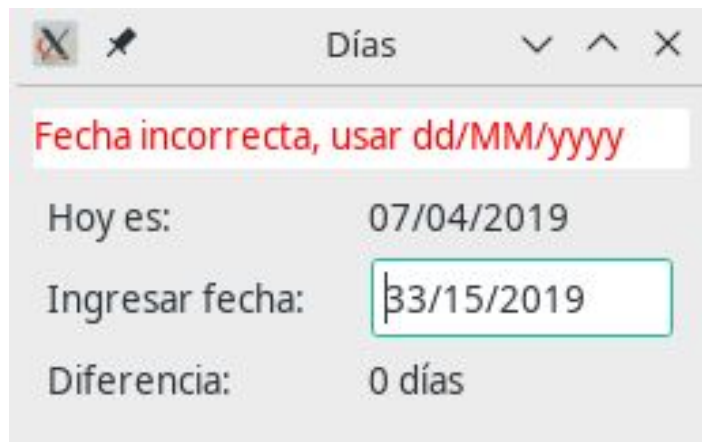
```
class DateBox(container: Panel) : TextBox(container) {  
    override fun <M, C: ControlBuilder>  
        bindValueToProperty(propertyName: String): Binding<M, Control, C> {  
        val binding = super.bindValueToProperty<M, C>(propertyName)  
        binding.setTransformer(DateTransformer())  
        withFilter(DateTextFilter())  
        return binding  
    }  
}  
  
class TimeCounterWindow : MainWindow<TimeCounter> {  
    override fun createContents(mainPanel: Panel) {  
        /* ... */  
        Label(mainPanel) withText "Ingresar fecha:"  
        DateBox(mainPanel) bindTo "date"  
    }  
}
```

Filter + Transformer



A screenshot of a web form titled "Días" with a search icon and a dropdown arrow. The form contains three rows: "Hoy es:" with the value "07/04/2019", "Ingresar fecha:" with a text input containing "09/04/2019", and "Diferencia:" with the value "2 días".

Hoy es:	07/04/2019
Ingresar fecha:	09/04/2019
Diferencia:	2 días



A screenshot of a web form titled "Días" with a search icon and a dropdown arrow. A red error message "Fecha incorrecta, usar dd/MM/yyyy" is displayed at the top. The form contains three rows: "Hoy es:" with the value "07/04/2019", "Ingresar fecha:" with a text input containing "03/15/2019", and "Diferencia:" with the value "0 días".

Fecha incorrecta, usar dd/MM/yyyy

Hoy es:	07/04/2019
Ingresar fecha:	03/15/2019
Diferencia:	0 días

No siempre se logra evitar el chequeo en el Transformer

Manejo de Colecciones

Existen varios componentes que permiten representar listados de datos:

- ▶ Selector (Dropdown)
- ▶ List
- ▶ Radio Selector
- ▶ Tables

Selectors

The image shows a web form window titled "Data". On the left, there is a vertical list of product categories: "Computadora» (\$30000)", "Teclado» (\$800)", "Mouse Inalámbrico» (\$700)", and "Notebook» (\$25000)". The "Computadora» (\$30000)" item is highlighted with a green background. To the right of this list is a dropdown menu currently showing "Mouse Inalámbrico» (\$700)". Below the dropdown is a group of four radio buttons, each corresponding to one of the product categories. The radio button for "Notebook» (\$25000)" is selected, indicated by a green dot in the center of the circle.

Computadora» (\$30000)
Teclado» (\$800)
Mouse Inalámbrico» (\$700)
Notebook» (\$25000)

Mouse Inalámbrico» (\$700) ▾

☐ Computadora» (\$30000)
☐ Teclado» (\$800)
☐ Mouse Inalámbrico» (\$700)
☒ Notebook» (\$25000)

Selectors \Rightarrow Adapters

Para mostrar un dato con formato complejo tenemos dos opciones:

- ▶ Redefinir el `toString()` del modelo (mala idea, ¿por qué?)
- ▶ Setear un *Adapter*

Adapters ⇒ Selectors

```
class StoreWindow(model: Store) :  
    MainWindow<Store>(model) {  
    override fun createContents(mainPanel: Panel) {  
        mainPanel.asHorizontal()  
        List<Product>(mainPanel) with {  
            bindTo("pc")  
            bindItemsTo("products")  
            .adaptWithProp<Product>("description")  
        }  
        Selector<Product>(mainPanel) with {  
            bindTo("mouse")  
            bindItemsTo("products")  
            .adaptWithProp<Product>("description")  
        }  
        RadioSelector<Product>(mainPanel) with {  
            bindTo("notebook")  
            bindItemsTo("products")  
            .adaptWithProp<Product>("description")  
        }  
    }  
}
```

```
@Observable class Store {  
    var products = listOf(  
        Product("Computadora", 30000),  
        Product("Teclado", 800),  
        Product("Mouse Inalámbrico", 700),  
        Product("Notebook", 25000)  
    )  
    var pc = products[0]  
    var mouse = products[2]  
    var notebook = products[3]  
}  
  
@Observable class Product  
    (var name:String, var price:Int) {  
    fun description()="$name» ($$price)"  
}
```

Tablas



Nombre	Duración
Tema de Pototo	3 min
Hablando de la Libertad	7 min
Light my Fire	6 min
Whisky in the jar	5 min

Tables ⇒ Columns

```
class PlaylistWindow : MainWindow<Playlist> {  
    override fun createContents(mainPanel: Panel) {  
        title = "Unquify"  
        table<Song>(mainPanel) {  
            bindTo("songs")  
            bindSelectionTo("selectedSong")  
            column {  
                title = "Nombre"  
                fixedSize = 200  
                bindContentsTo("name")  
            }  
            column {  
                title = "Duración"  
                fixedSize = 150  
                bindContentsTo("durDescrip")  
            }  
        }  
    }  
}
```

```
@Observable class Playlist {  
    val songs = listOf(  
        Song("Tema de Pototo", 3),  
        Song("Hablando de la Libertad", 7),  
        Song("Light my Fire", 6),  
        Song("Whisky in the jar", 5)  
    )  
    var selectedSong = songs[0]  
}  
  
@Observable class Song {  
    constructor(val name, val duration)  
    fun durDescrip()="$duration min"  
}
```

¿Demo?

