



Universidad
Nacional
de Quilmes

***TRANSFORMERS
FILTERS
ADAPTERS***

CONSTRUCCIÓN DE INTERFACES DE USUARIO

1er Cuatrimestre 2019

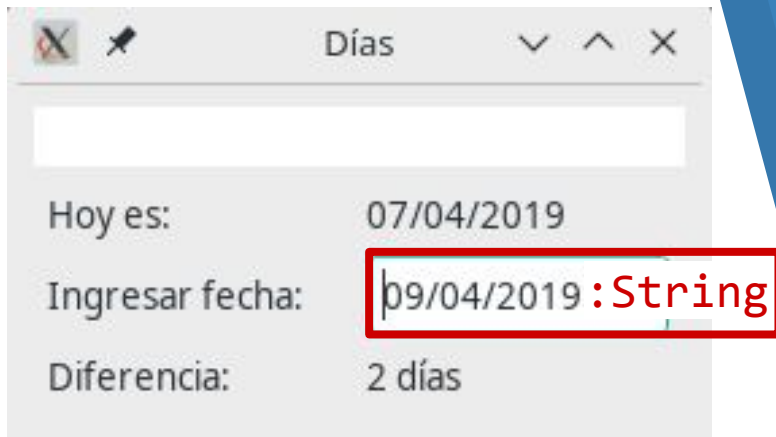


Problemas de *Binding*

Modelo

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

Vista



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 rectangular box, indicating a binding error where the input is treated as a String instead of a LocalDate.

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

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.cor  
e.databinding -  
    0 - Could not  
change value of  
ar.unq.edu.ui.a  
rena.ej_contado  
r_dias.TimeCoun  
ter@6850b758  
    .another  
    Java.lang.  
IllegalArgumentException  
Exception:  
argument type  
mismatch
```

¿Cómo se resuelve?

Utilizando un objeto que transforme:

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

O sea, un *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)  
        }  
    }  
}
```

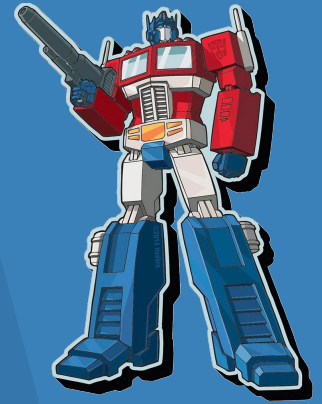


Transformers ⇒ Uso

```
class TimeCounterWindow : MainWindow<TimeCounter> {  
    constructor(model: TimeCounter) : super(model)  
    override fun createContents(mainPanel: Panel) {  
        title = "Días"  
        /*...*/  
  
        Label(datePanel).setText("Ingresar fecha:")  
        TextBox(datePanel)  
            .bindValueToProperty<Any, ControlBuilder>("date")  
            .setTransformer(DateTransformer())  
  
        Label(datePanel).setText("Diferencia:")  
        Label(datePanel)  
            .bindValueToProperty<Any, ControlBuilder>("diffStr")  
    }  
}
```



Transformers \Rightarrow Resultado



Días

Hoy es: 07/04/2019

Ingresar fecha: 09/04/2019

Diferencia: 2 días

Días

Fecha incorrecta, usar dd/MM/yyyy

Hoy es: 07/04/2019

Ingresar fecha: xx/04/2019

Diferencia: 2 días

Problemas de *Transformers*

- ▶ 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?
- ▶ El Transformer tiene más responsabilidad de la necesaria

¿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 = listOf(  
            "\\d", "\\d?", "/", "\\d", "\\d?", "/", "\\d{0,4}"  
        )  
        val regex = expected.reversed()  
            .fold("") { accum, elem -> "($elem$accum)?" }  
            .toRegex()  
        return event.potentialTextResult.matches(regex)  
    }  
}
```

Regex Resultante

`(\d(\d?(/(\d(\d?(/(\d{0,4})?)?)?)?)?)?)?)?`

01/12/2018

01/12/201

01/12/20

01/12/2

01/12/

01/12

01/1

01/

01

1

Filters ⇒ Uso directo

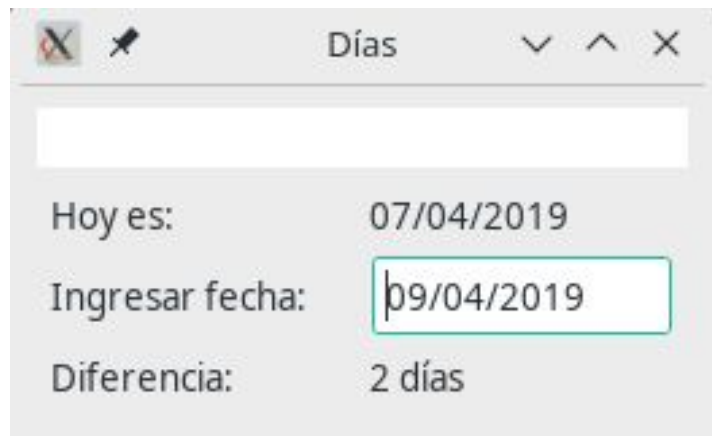
```
class TimeCounterWindow : MainWindow<TimeCounter> {  
    constructor(model: TimeCounter) : super(model)  
    override fun createContents(mainPanel: Panel) {  
        title = "Días"  
        /*...*/  
        Label(datePanel).setText("Ingresar fecha:")  
        TextBox(datePanel)  
            .withFilter(DateTextFilter())  
            .bindValueToProperty<Any, ControlBuilder>("date")  
            .setTransformer(DateTransformer())  
        /*...*/  
    }  
}
```

Filters \Rightarrow 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())
        this.withFilter(DateTextFilter())
        return binding
    }
}

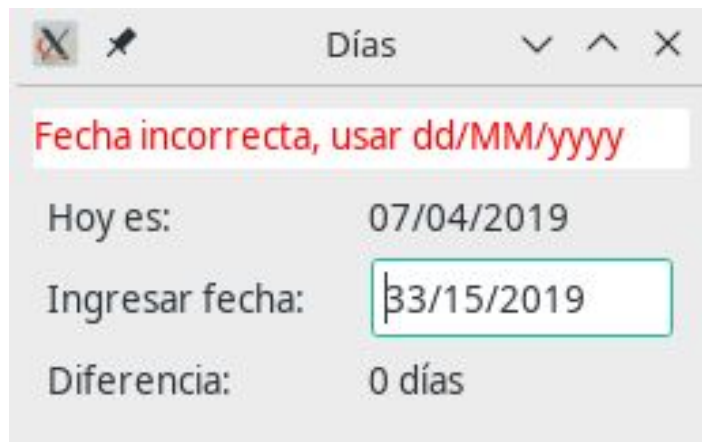
class TimeCounterWindow : MainWindow<TimeCounter> {
    override fun createContents(mainPanel: Panel) {
        title = "Días"
        /*...*/
        Label(datePanel).setText("Ingresar fecha:")
        DateBox(datePanel).bindValueToProperty<Any, ControlBuilder>("date")
        /*...*/
    }
}
```

Filter + Transformer



A screenshot of a web application window titled "Días". It features a search bar at the top. Below it, the text "Hoy es:" is followed by the date "07/04/2019". The label "Ingresar fecha:" is followed by a text input field containing "09/04/2019". The label "Diferencia:" is followed by the text "2 días".

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



A screenshot of a web application window titled "Días". It features a search bar at the top. Below it, the text "Hoy es:" is followed by the date "07/04/2019". The label "Ingresar fecha:" is followed by a text input field containing "3/15/2019". The label "Diferencia:" is followed by the text "0 días". A red error message "Fecha incorrecta, usar dd/MM/yyyy" is displayed above the input field.

Fecha incorrecta, usar dd/MM/yyyy

Hoy es:	07/04/2019
Ingresar fecha:	3/15/2019
Diferencia:	0 días

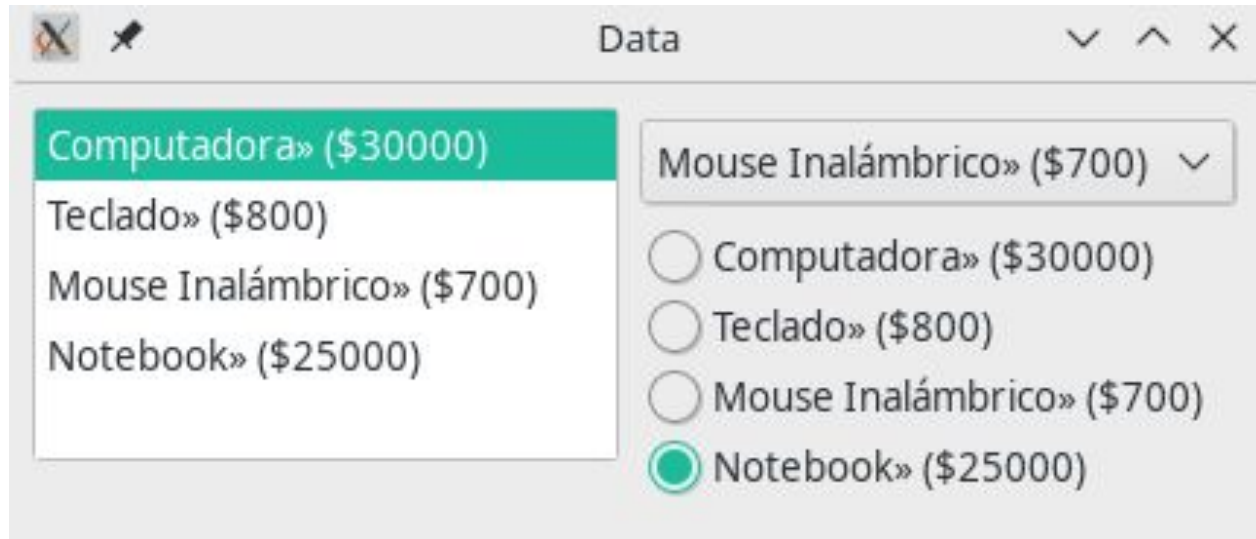
No siempre se logra evitar el chequeo en el Transformer

Manejo de Colecciones

Existen varios componentes que permiten representar listados de datos:

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

Selectors



The image shows a window titled "Data" with a standard OS title bar (minimize, maximize, close buttons). Inside the window, there is a list of items on the left and a selection area on the right. The list on the left contains four items: "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 radio button selection area with four options: "Computadora» (\$30000)", "Teclado» (\$800)", "Mouse Inalámbrico» (\$700)", and "Notebook» (\$25000)". The "Notebook» (\$25000)" option is selected, indicated by a green radio button.

Item	Price
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 : MainWindow<Store> {  
    override fun createContents(mainPanel: Panel) {  
        mainPanel.setLayout(HorizontalLayout())  
        val prodList = List<Product>(mainPanel)  
        prodList.bindValueToProperty<...>("pc")  
        prodList.bindItemsToProperty("products")  
            .adaptWith(Product::class.java, "description")  
  
        val rightPanel = Panel(mainPanel)  
        val prodSelect = Selector<Product>(rightPanel)  
        prodSelect.bindValueToProperty<...>("mouse")  
        prodSelect.bindItemsToProperty("products")  
            .adaptWith(Product::class.java, "description")  
  
        val prodRadio = RadioSelector<Product>(rightPanel)  
        prodRadio.setWidth(200)  
        prodRadio.bindValueToProperty<...>("notebook")  
        prodRadio.bindItemsToProperty("products")  
            .adaptWith(Product::class.java, "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



A screenshot of a software window titled "Unquify". The window contains a table with two columns: "Nombre" (Name) and "Duración" (Duration). The table has five rows of data. The first row, "Tema de Pototo", is highlighted in green. The other rows are "Hablando de la Libertad", "Light my Fire", and "Whisky in the jar". The window has standard OS controls (minimize, maximize, close) in the top right corner.

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 \Rightarrow Columns

```
class PlaylistWindow : MainWindow<Playlist> {  
    override fun createContents(mainPanel: Panel) {  
        title = "Unquify"  
        val songs = Table(mainPanel, Song::class.java)  
        songs.setNumberVisibleRows(5)  
        songs.bindItemsToProperty("songs")  
        songs.bindValueToProperty<...>("selectedSong")  
        Column<Song>(songs)  
            .setTitle("Nombre")  
            .setFixedSize(200)  
            .bindContentsToProperty("name")  
        Column<Song>(songs)  
            .setTitle("Duración")  
            .setFixedSize(150)  
            .bindContentsToProperty("durationDescription")  
    }  
}
```

```
@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  
(val name: String, val duration: Int) {  
    fun durationDescription() =  
        "$duration min"  
}
```

¿Demo?

