



Universidad  
Nacional  
de Quilmes

# DISEÑO Y NAVEGACIÓN

*LAYOUTS, WINDOWS  
& DIALOGS*

# CONSTRUCCIÓN DE INTERFACES DE USUARIO

1er Cuatrimestre 2019



# Layout

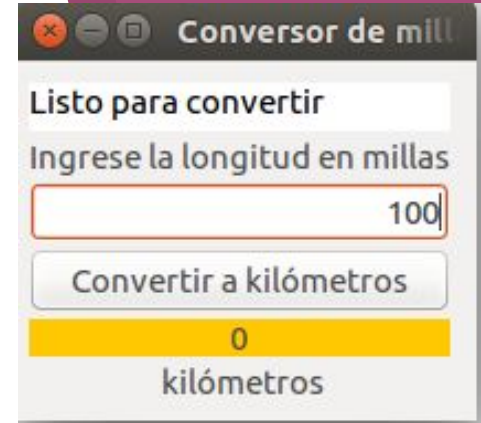
- ▶ En español: Diseño o Disposición
- ▶ Define **cómo** se van a acomodar los componentes visuales.
- ▶ No es un componente visual (no se lo puede “ver” directamente)

# Tipos de Layout

- ▶ `VerticalLayout`
- ▶ `HorizontalLayout`
- ▶ `ColumnLayout`

# VerticalLayout

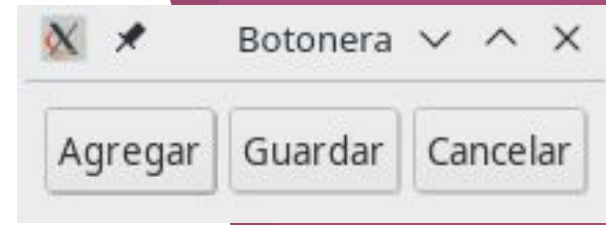
- ▶ Los componentes se disponen verticalmente.
- ▶ O sea uno debajo del otro.
- ▶ Es la disposición por defecto.
- ▶ Muy útil para:
  - Ventanas simples
  - Paneles de lado



# VerticalLayout ⇒ Código

```
class GeoWindow : MainWindow<GeoModel> {  
  
    constructor(model: GeoModel) : super(model)  
  
    override fun createContents(mainPanel: Panel) {  
        title = "Geo con Layouts"  
  
        mainPanel.setLayout(VerticalLayout())  
  
        /* ... Resto del Código... */  
    }  
}
```

# HorizontalLayout



- ▶ Los componentes se crean uno al lado del otro
- ▶ De izquierda a derecha
- ▶ Muy útil para:
  - Botoneras
  - Columnas de tamaño variable

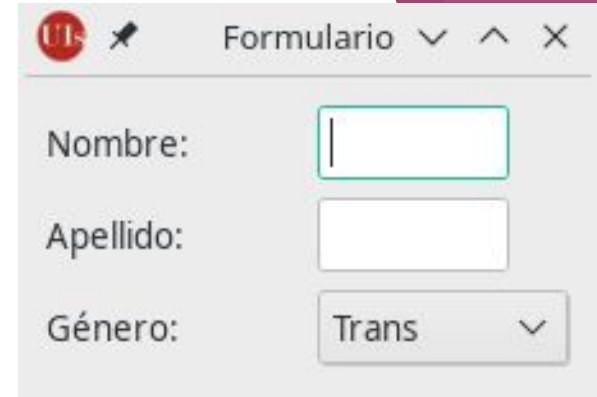
# HorizontalLayout ⇒ Código

```
class KeypadWindow : MainWindow<Keypad> {  
    constructor(model: Keypad) : super(model)  
  
    override fun createContents(mainPanel: Panel) {  
        title = "Botonera"  
  
        mainPanel.setLayout(HorizontalLayout())  
  
        Button(mainPanel).setCaption("Agregar")  
        Button(mainPanel).setCaption("Guardar")  
        Button(mainPanel).setCaption("Cancelar")  
    }  
}
```



# ColumnLayout

- ▶ Agrupa los componentes en columnas.
- ▶ Se debe indicar la cantidad de columnas.
- ▶ Los componentes se crean uno al lado del otro (de izq a der) hasta completar la fila.
- ▶ Luego pasan a la siguiente fila.
- ▶ Muy útil para:
  - Formularios
  - Columnas de igual tamaño

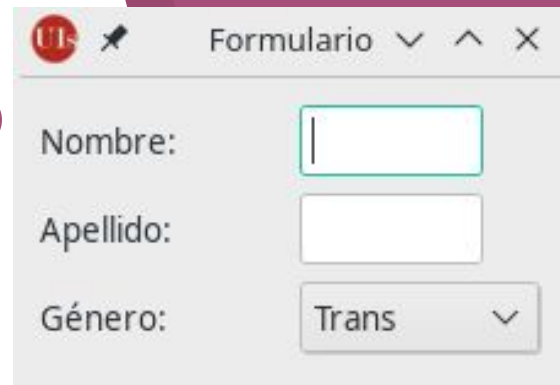


The screenshot shows a Java Swing window titled "Formulario" with a standard Mac OS X title bar (red, yellow, green buttons). The window contains a form with three labels and input fields arranged vertically. The first label is "Nombre:" followed by a text input field. The second label is "Apellido:" followed by a text input field. The third label is "Género:" followed by a dropdown menu showing "Trans" with a downward arrow. The components are aligned to the left, and the input fields are of equal width, demonstrating the use of ColumnLayout.



# ColumnLayout ⇒ Código




```
class FormWindow(model: Form) : MainWindow<Form>(model) {  
    override fun createContents(mainPanel: Panel) {  
        title = "Formulario"  
        mainPanel.setLayout(ColumnLayout(2))  
  
        Label(mainPanel).setText("Nombre:"); TextBox(mainPanel)  
        Label(mainPanel).setText("Apellido:"); TextBox(mainPanel)  
  
        Label(mainPanel).setText("Género:")  
        val selector = Selector<String>(mainPanel)  
        selector.bindValueToProperty<String, ControlBuilder>("genre")  
        selector.bindItems<String>(ObservableProperty(model, "genresList"))  
    }  
}  
  
@Observable class Form {  
    var genresList = listOf("Masculino", "Femenino", "Trans", "Otro")  
    var genre = "Trans"  
}
```



The screenshot shows a JavaFX window titled "Formulario". It contains three labels and their corresponding controls:

- "Nombre:" followed by a text input field.
- "Apellido:" followed by a text input field.
- "Género:" followed by a dropdown menu with "Trans" selected and a downward arrow.

# Mezclando Layouts

-  Horizontal
-  Vertical
-  Columnas

TV Series

Breaking Bad

Lost

The Walking Dead

New

Name: Breaking Bad

Seasons: 5

Actor	Character	
Bryan Cranston	Walter White	Delete
Aaron Paul	Jesse Pinkman	Delete
Bob Odenkirk	Saul Goodman	Delete

Ok

Cancel

Add Actor

**¿Preguntas  
hasta acá?**



# NAVEGACIÓN

*WINDOWS  
& DIALOGS*



# Ventana Principal

Hasta el momento venimos usando una sola ventana:

- ▶ Creamos una `LoQueSeaWindow` que extienda de `MainWindow<T>`.
- ▶ Definimos el constructor pasándole el modelo.
- ▶ Sobreescribimos el método `createContents` con el contenido.
- ▶ Y levantamos la aplicación desde el `main`.

# Ventana Principal

```
class MainCompanyWindow : MainWindow<Company> {  
    constructor(model: Company) : super(model)  
    override fun createContents(mainPanel: Panel) {  
        title = "Empresa"  
        Label(mainPanel).setText("Empleados: ")  
        val employees = List<Employee>(mainPanel)  
        employees.bindValueToProperty<Employee, ControlBuilder>("empl")  
        employees.bindItemsToProperty("employees")  
    }  
}  
  
@Observable class Company {  
    var empl: Employee? = null  
    var employees = listOf(Employee("Jon Snow"), ...)  
}  
  
@Observable class Employee(private val name: String) {  
    override fun toString() = name  
}
```



# ¿Y cómo levanto otra ventana?

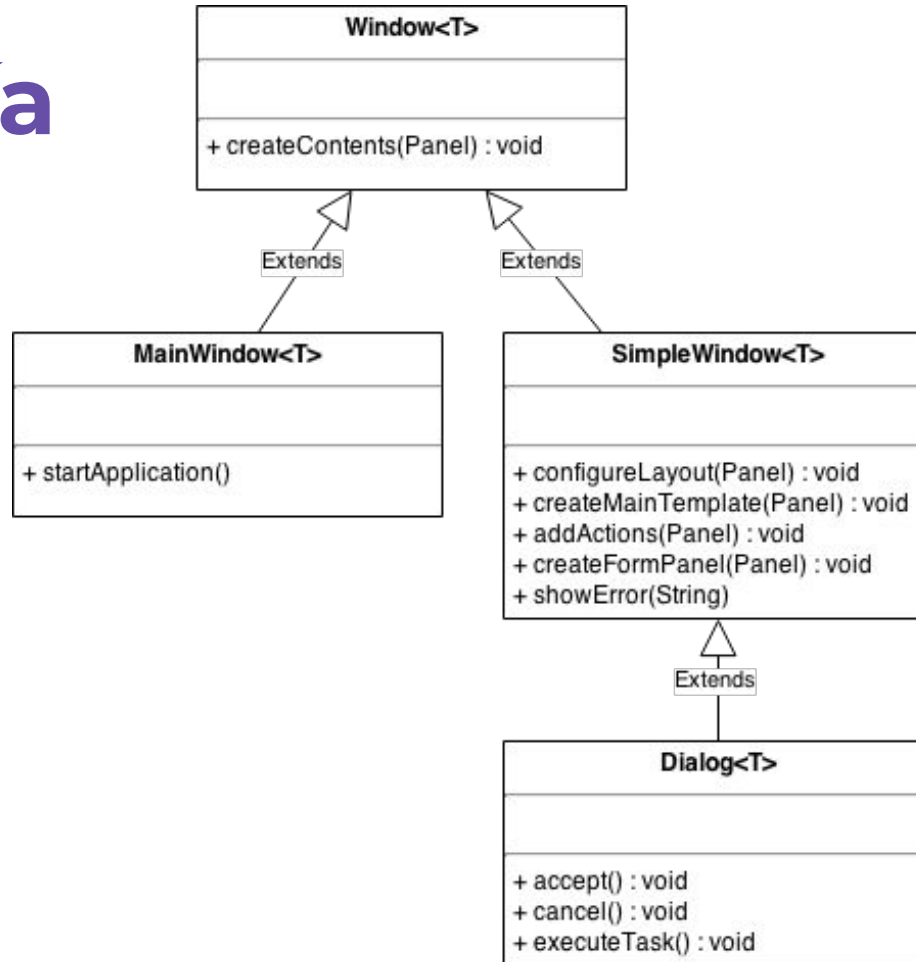
- ▶ A partir de algún evento podemos *instanciar* una nueva ventana.
- ▶ Esta ventana debe conocer a la ventana que la invocó (solemos llamarla *owner* o *parent*).
- ▶ También debe tener un *modelObject* que puede ser “pasado” desde el parent.

# Tipos de Ventanas

- ▶ **Window:** es la clase abstracta para todas las ventanas.
- ▶ **MainWindow:** es un tipo especial de ventana que se usa para aplicaciones simples o de una sola ventana.
- ▶ **SimpleWindow:** ventana común que agrega el panel de errores.
- ▶ **Dialog:** es una ventana final que depende de alguna de las anteriores y que debe generar una acción y cerrarse.



# Jerarquía

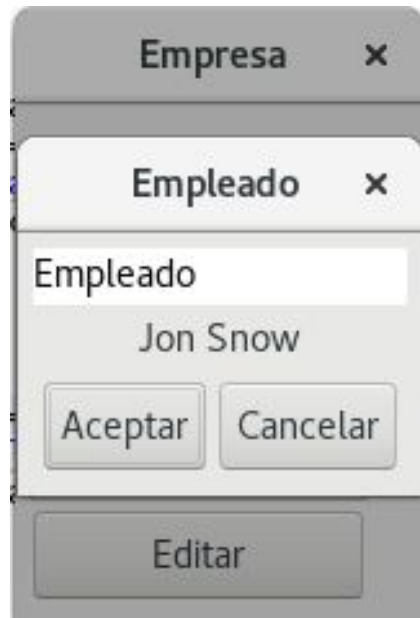


# Window $\Rightarrow$ Dialog

```
class MainCompanyWindow : MainWindow<Company> {  
    override fun createContents(mainPanel: Panel) {  
        /* ... */  
        // Abro Dialog  
        Button(mainPanel)  
            .setCaption("Agregar White Walker")  
            .onClick { edit() }  
    }  
    private fun edit() {  
        val dialog = EmployeeDialog(this, modelObject.employees[0])  
        dialog.onAccept {  
            modelObject.employees.add(Employee("White Walker"))  
        }  
        dialog.onCancel { /* Do Nothing */ }  
        dialog.open()  
    }  
}
```

# Dialog

```
class EmployeeDialog : Dialog<Employee> {  
    constructor(owner WindowOwner, model: Employee) : super(owner, model)  
  
    override fun addActions(actions: Panel) {  
        Button(actions)  
            .setCaption("Aceptar")  
            .onClick { accept() }  
        Button(actions)  
            .setCaption("Cancelar")  
            .onClick { cancel() }  
    }  
  
    override fun createFormPanel(mainPanel: Panel) {  
        Label(mainPanel).setText(this.modelObject.name)  
    }  
}
```



# Window $\Rightarrow$ Window

- ▶ Las ventanas `Dialog` funcionan como “modales”, o sea que se usan para una función específica y se cierran.
- ▶ Se pueden seguir abriendo ventanas o dialogs desde un `Dialog` pero no es recomendable porque se van “stackeando”.
- ▶ Para poder trabajar con ventanas independientes deben ser `Window` o `SimpleWindow`.
- ▶ Una `MainWindow` puede abrir una `Window` pero luego nunca más se puede volver a la `MainWindow`.

# Window $\Rightarrow$ Window

- ▶ Pero las ventanas `Window` no pueden ser inicializadas desde un `main()` como sí sucedía con `MainWindow`
- ▶ Es necesario otra estrategia de inicialización
  - ▷ Hay que usar la clase `Application`
  - ▷ Que se encarga de inicializar la aplicación y llamar a la `Window` que indiquemos como “inicial”
  - ▷ Luego vamos a poder interactuar entre `Windows` y `Dialogs` libremente

# Application ⇒ Window

```
fun main() = MyApplication().start()
```

```
class MyApplication : Application() {  
    override fun createMainWindow(): Window<*> {  
        CompanyWindow(this, Company())  
    }  
}
```

# Window ⇔ Window

```
class CompanyWindow : SimpleWindow<Company> {
    constructor(parent: WindowOwner, model: Company) {
        super(parent, model)
    }
    override fun addActions(actionsPanel: Panel) {}
    override fun createFormPanel(mainPanel: Panel) {}
    override fun createContents(mainPanel: Panel) {
        title = "Empresa"
        Button(mainPanel)
            .setCaption("Abrir Empleado")
            .onClick {
                close()
                EmployeeWindow(
                    this,
                    modelObject.employees[0]
                ).open()
            }
    }
}
```

```
class EmployeeWindow : SimpleWindow<Employee> {
    constructor(owner: WindowOwner, model: Employee) {
        super(owner, model)
    }
    override fun addActions(actionsPanel: Panel) {}
    override fun createFormPanel(mainPanel: Panel) {}
    override fun createContents(mainPanel: Panel) {
        title = "Empleado"
        Button(mainPanel)
            .setCaption("Volver a Empresa")
            .onClick {
                close()
                CompanyWindow(
                    this,
                    Company()
                ).open()
            }
    }
}
```

# Demo

