



Universidad
Nacional
de Quilmes

CONSTRUCCIÓN DE INTERFACES DE USUARIO

1er Cuatrimestre de 2019

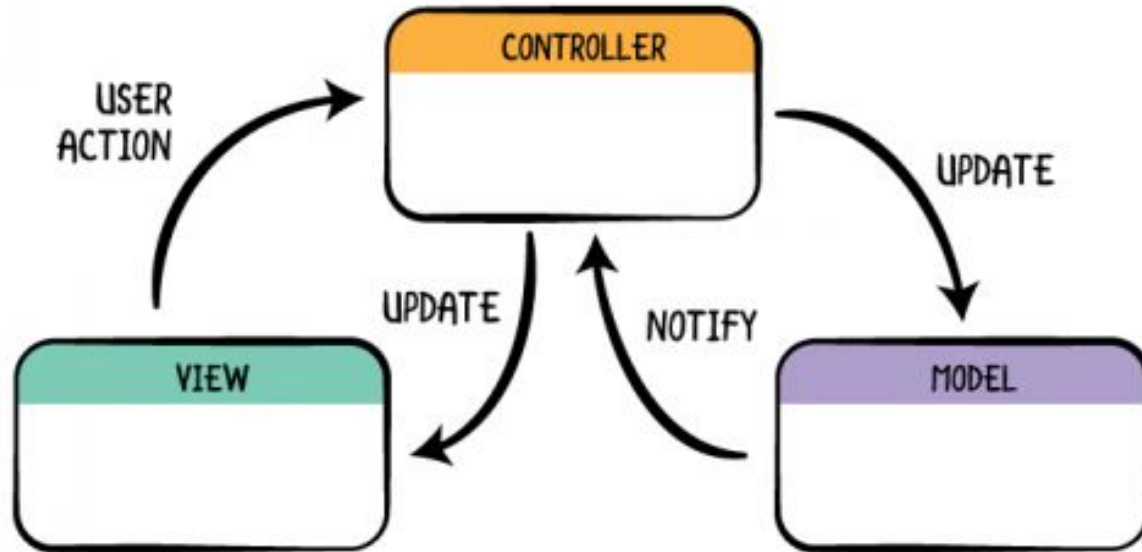


Repaso Binding y Eventos

- ▶ Eventos
 - ▷ Tanto la vista como el modelo producen eventos.
 - ▷ Inversión relación de conocimiento
 - ▷ Similar patron Observer
- ▶ Binding
 - ▷ Muchos frameworks proveen mecanismos de binding
 - ▷ Relaciona dos variables manteniendolas vinculadas
 - ▷ Unidireccional y Bidireccional

Patrón Model-View-Controller

Patrón de arquitectura



Patrón Model-View-Controller

Patrón de diseño o modelo de abstracción utilizado para definir y estructurar los componentes necesarios en el desarrollo de software.

Patrón Model-View-Controller

- ▶ Vista
 - ▷ Capa de presentación / interfaz de usuario
 - ▷ Ventanas, botones, inputs.

Patrón Model-View-Controller

- ▶ Modelo
 - ▷ Capa que contiene la lógica de la aplicación
 - ▷ Contiene la lógica de negocio/reglas de negocio
 - ▷ Modelo de objetos
 - ▷ Ejemplo: objetos que representen:
 - ▷ Un contacto de una agenda
 - ▷ Un movimiento de una caja de ahorro.

Patrón Model-View-Controller

- ▶ Controlador
 - ▷ Intermediario entre la capas Vista y Modelo.
 - ▷ Orquestadores del modelo o decisores de la operación
 - ▷ Gestiona el flujo de información entre el las capas.
 - ▷ Adapta la vista con el modelo.

Patrón Model-View-Controller

- ▶ Modelo - Errores comunes
 - ▷ Aparición de lógica de negocio en otras capas
 - ▷ Modelo con elementos propios de interacción con usuario

Patrón Model-View-ViewModel MVVM

- ▶ Arena es un framework que utiliza este tipo de patrón
- ▶ Vista: Similar a MVC
- ▶ Modelo: Similar a MVC
- ▶ ViewModel: Implementación de Patrón Observer, en Arena a través del mecanismo de Binding.

Patrón Model-View-ViewModel

► Ejemplo Arena - Login - View

```
class LoginWindow : MainWindow<Login>(Login()) {  
    override fun createContents(panel: Panel) {  
  
        Label(panel).setText("Ingreso el usuario")  
  
        TextBox(panel).bindValueToProperty<Int, ControlBuilder>("user")  
  
        Label(panel).setText("Ingreso el password")  
  
        PasswordField(panel).bindValueToProperty<Int, ControlBuilder>("password")  
  
        Button(panel)  
            .setCaption("Autenticar")  
            .onClick { modelObject.authenticate() }  
  
        val status = Label(panel)  
        status.bindValueToProperty<Double, ControlBuilder>("authenticated")  
        status.bindBackgroundToProperty<ControlBuilder, Any, Any>("authenticatedColor")  
    }  
}
```

Patrón Model-View-ViewModel

► Ejemplo Arena - Login - Modelo

```
@Observable
class Login {
    var user: String = ""
        set(value) {
            resetAuth()
            field = value
        }
    var password: String = ""
        set(value) {
            resetAuth()
            field = value
        }
    var authenticated: Boolean = false
    var authenticatedColor: Color = Color.ORANGE

    fun authenticate() {
        authenticated = (user == "polo") && (password == "polo")
        authenticatedColor = if (authenticated) Color.GREEN else Color.ORANGE
    }

    private fun resetAuth() {
        authenticated = false
        authenticatedColor = Color.ORANGE
    }
}
```

Patrón Model-View-ViewModel

► Ejemplo Arena - Login - ViewModel

```
class LoginWindow : MainWindow<Login>(Login()) {  
    override fun createContents(panel: Panel) {
```

```
        this.title = "Login del sistema"
```

```
        Label(panel).setText("Ingreso el usuario")
```

```
        TextBox(panel).bindValueToProperty<Int, ControlBuilder>("user")
```

```
        Label(panel).setText("Ingreso el password")
```

```
        PasswordField(panel).bindValueToProperty<Int, ControlBuilder>("password")
```

```
        Button(panel)  
            .setCaption("Autenticar")  
            .onClick { modelObject.authenticate() }
```

```
        val status = Label(panel)
```

```
        status.bindValueToProperty<Double, ControlBuilder>("authenticated")
```

```
        status.bindBackgroundToProperty<ControlBuilder, Any, Any>("authenticatedColor")
```

```
    }
```

```
@Observable
```

```
class Login {
```

```
    var user: String = ""
```

```
    set(value) {  
        resetAuth()  
        field = value
```

```
    }
```

```
    var password: String = ""
```

```
    set(value) {  
        resetAuth()  
        field = value
```

```
    }
```

Patrón MVC o MVVC

- ▶ Beneficios
 - ▷ Reutilización de código
 - ▷ Separación de conceptos - Legibilidad
 - ▷ Facilitar el desacople para el desarrollo
 - ▷ Facilitar el mantenimiento
 - ▷ Aparición de perfiles especializados en cada capas
 - ▷ Desacople de tecnologías
 - ▷ Mantenimiento - Testing

Application Model vs Model Simple

Cada vista necesita un modelo pero cuando la complejidad de interacción entre el usuario y el modelo crece el Modelo Simple no se ajusta bien por lo que aparece el concepto de Application Model

Consiste en crear un objeto que tenga la representación del comportamiento global de la aplicación. Generalmente representan un Caso de Uso

Application Model vs Model Simple

Diferentes Ciclos de vida

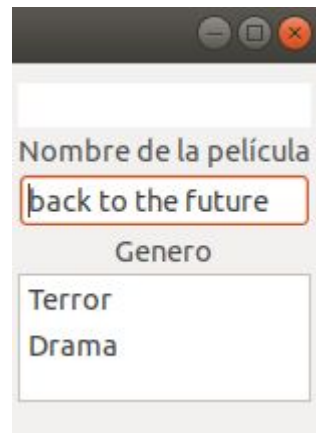
Los objetos de dominio se crear, se persisten, se modifican, etc.

Los objetos de aplicación se usan una sola vez.

Application Model vs Model Simple - Ejemplo

```
class Gender{  
    var name : String = ""  
    constructor(name: String) {  
        this.name = name  
    }  
    override fun toString(): String {  
        return this.name  
    }  
}
```

```
class Film {  
  
    var name: String = ""  
    override fun toString(): String {  
        return this.name  
    }  
}
```



Nombre de la película

back to the future

Genero

Terror

Drama

Application Model vs Model Simple - Ejemplo

@Observable

```
class FilmCreateAppModel{
```

```
    var film : Film = Film()
```

```
    var genders : MutableList<Gender> = ArrayList<Gender>()
```

```
}
```

```
class Window : SimpleWindow<FilmCreateAppModel> {
```

```
    constructor (owner: WindowOwner, model: FilmCreateAppModel) : super(owner, model)
```

```
    override fun createFormPanel(mainPanel: Panel) {
```

```
        Label(mainPanel).setText("Nombre de la película")
```

```
        TextBox(mainPanel).bindValueToProperty<String, ControlBuilder>("film")
```

```
        Label(mainPanel).setText("Genero")
```

```
        List<FilmCreateAppModel>(mainPanel).bindItemsToProperty("genders")
```

```
    }
```

```
}
```

¿Preguntas?