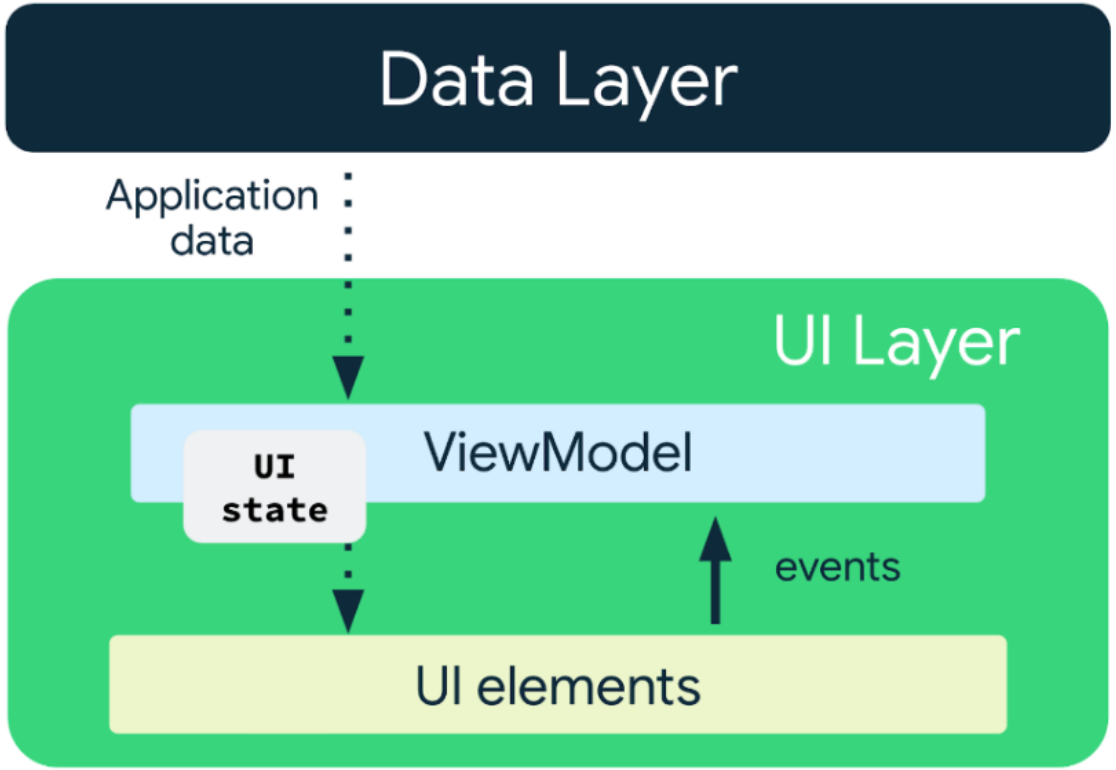
Software para Dispositivos Móviles

Miguel Sánchez Santillán
sanchezsmiguel@uniovi.es

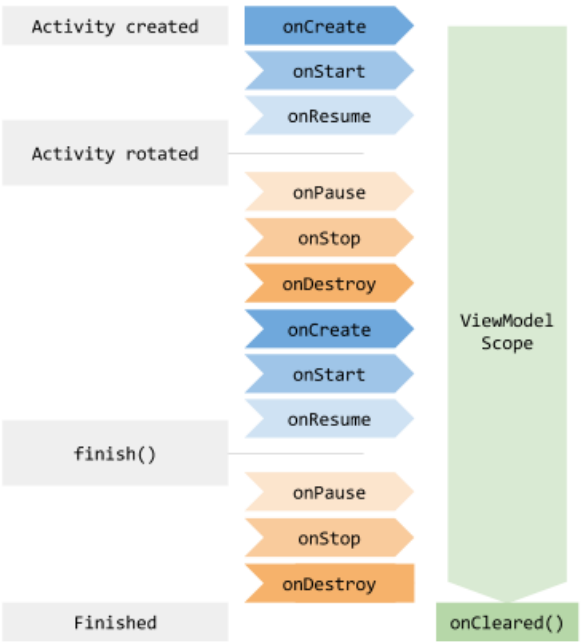
MVVM – Model–View-ViewModel

- **Modelo**: Datos / Negocio.
- **View**: Vista y relacionados.
  - Componentes.
  - Eventos.
- **ViewModel**: Estado de la Vista.
  - **Notifica** a View los cambios.
  - Porque View está **observando**.



ViewModel – Estado de la Interfaz

- Almacena y expone el **estado de la interfaz**.
  - Accede a/o proporciona la lógica empresarial.
- Sobrevive a los **cambios de configuración** →
- Pueden vincularse a:
  - Una **Activity**: Hasta que ésta finalice.
  - Un **Fragment**: Hasta que se desvincule.
  - Elementos **Navigation**: Un grafo, por ejemplo.



Ciclo: Actividad vs ViewModel

Ejercicio I – El Número Aleatorio

- Crea un proyecto vacío, como siempre.
  - Opcional: Activa el **view binding** en el build.gradle

- Implementa una interfaz con un **TextView** y un **Button**.

- Crea la lógica de la aplicación, como siempre.
  - **Al pulsar el botón, muestra en el TextView un aleatorio en [0, 10]**

- **Este valor se perderá** cuando se dé un cambio de configuración.

Ejercicio II – Añadir un ViewModel

- Una clase es un ViewModel si **extiende de ViewModel**.
  - La clase se llamará MainActivityViewModel

- La clase **MainActivity** tendrá un **atributo viewModel**.
  - El tipo de éste será el de la clase creada anteriormente.

- El ViewModel está vinculado al ciclo de vida.
  - Y por ello, **no lo podemos inicializar invocando al constructor**.
  - Se inicializa con **by viewModels()**

LiveData – Notificar y Observar

- Es un contenedor de datos **observable**.
  - Tenemos **LiveData** y **MutableLiveData**.

- Podemos acceder al valor con variable**.value**
  - Siendo variable de tipo LiveData/MutableLiveData

- Los valores pueden ser modificados (MutableLiveData) con:
  - variable.**value =** nuevoValor → Si **estamos en el hilo principal**
  - variable.**postValue**(nuevoValor) → Si **estamos en otro hilo**.
  - Ambas opciones desencadenarán una notificación a los observadores.

Ejercicio III – Usar el ViewModel y LiveData

- Se crean en el ViewModel y se observan desde la Interfaz.

```
private val _numero = MutableLiveData( value: -1)
val numero: LiveData<Int>
    get() = _numero
```

```
viewModel.numero.observe( owner: this) { newNumero ->
    //Código
}
```

- Crea una función en el ViewModel para crear el número aleatorio.
  - No lo devuelve, simplemente **actualiza el valor de _numero**.

- Al cambiar el valor y estar siendo observado, lo recibirás en la interfaz.
  - Completa el código para actualizar el valor del TextView

Ejercicio IV – ¿Funciona?

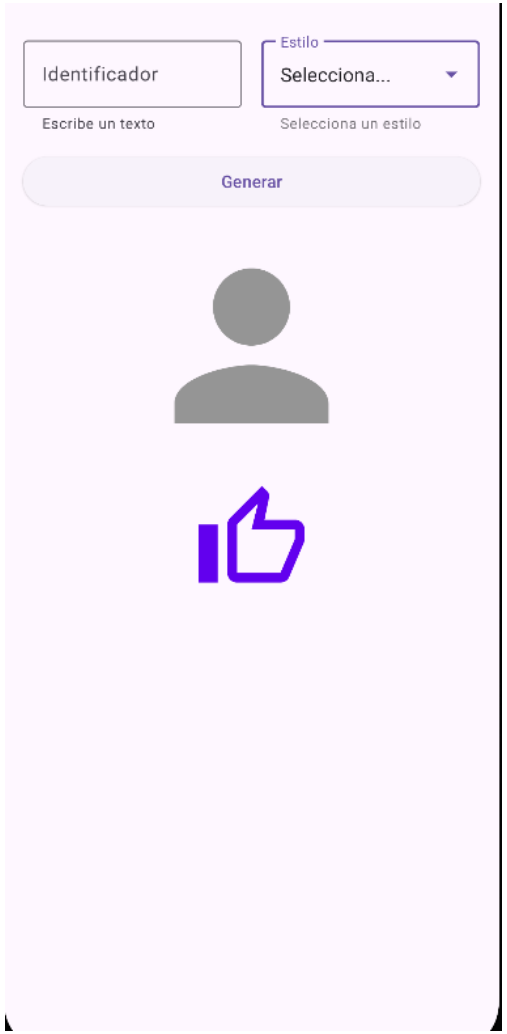
- Arranca la aplicación.
- No generes ningún número.
- Rota la pantalla.
- **Discurre e implementa un mecanismo para evitarlo**.

Ejercicio V – Creación de Avatares

- Proyecto zip de material complementario.

- Proyecto para generar Avatares.
  - Dada una cadena y un estilo, obtiene la imagen.
  - Se puede añadir a una lista de favoritos.
  - También se puede eliminar de la lista de favoritos.

- Completa primero la lógica.
  - **Sin ViewModel.**
  - **Sin LiveData.**

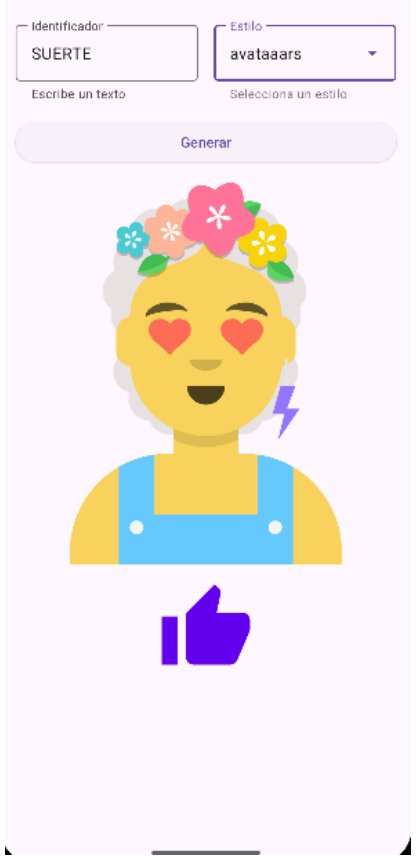


Ejercicio VI – ViewModel + LiveData

- Exporta el proyecto anterior.
  - A modo de backup.

- Implementa una nueva versión:
  - Emplea un ViewModel.
  - Emplea LiveData.

- **¿Cuál es el problema? ¿Posible solución/es?**



ViewModel – Parámetros en el constructor

- Para pasar **parámetros a un ViewModel durante su creación** se utiliza una **factoría**.

- Ej. con unparámetro (estilos):

```
class MainActivityViewModelProviderFactory(
    val estilos: Array<String> //Y el resto de parámetros necesarios
) : ViewModelProvider.Factory
{
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        //Aquí se invoca al constructor con los parámetros necesarios
        return MainActivityViewModel(estilos) as T
    }
}
```

- Atributo de la Activity:

```
private lateinit var viewModel : MainActivityViewModel
```

- En el onCreate():

```
viewModel = ViewModelProvider( owner: this,
    MainActivityViewModelProviderFactory(estilos)).get(MainActivityViewModel::class.java)
```

Ejercicio VII – Aplicación contactos

- Ve al Campus Virtual al apartado de la práctica 5.
- Descarga el proyecto: Aplicación contactos con DB [Solucionada]
- Intenta migrarla a MVVM.
- Recuerda la transparencia anterior.