

# **Software para Dispositivos Móviles**

Miguel Sánchez Santillán  
sanchezsmiguel@uniovi.es

# Application – Una clase global

- Las clases derivadas de Application mantienen un **estado global**.
- Útil para establecer dependencias: Room, Retrofit...
- Se deben apoyar en el uso de interfaces ¿**Por qué?**

# Uso de dependencias

- Añade la clase PeliculasApp incluida en el material complementario.
- Añádela al manifiesto:

```
<application  
    android:name=".PeliculasApp"
```
- **Actualiza TopPeliculasFragment para que la utilice.**

# Repositorio – Room + Retrofit

- ¿Cómo implementarías un repositorio en esta aplicación?
- Necesitas mantenerlo de forma global.
- Y recibirá tanto el objeto de Retrofit como el de Room.

# Repositorio II – Migrando funcionalidad

- Hay una serie de operaciones que deberían ser **métodos del repositorio**:
  - Obtener las películas populares.
  - Obtener una película dada una id.
  - Insertar una peliculaEntity a favoritas.
  - Eliminar una películaEntity de favoritas.
  - Obtener las películas favoritas.
- Implementa estas operaciones basándote en el código de la app.

# ViewModels – Paso de parámetros

- La idea es que **cada Fragmento tenga su propio ViewModel**.
  - Cada ViewModel recibirá el repositorio.
  - En la siguiente transparencia tienes el código aproximado.
- Empieza por el de TopPelículasFragment.
  - **Corrutinas: `viewModelScope`**
  - El Adapter quizás necesite un método para recibir una nueva lista.
    - Dentro debería notificar que los datos han cambiado.

# ViewModel – Parámetros en el constructor

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

- Ej. con un parámetro (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: `private lateinit var viewModel : MainActivityViewModel`

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

# ¿Por qué hay dos adapters?


- A la UI deberían llegar objetos con la información necesaria.
- Se están enviando dos **identidades diferentes**:
  - MovieResponse.
  - PeliculaEntity.
- **Se usa la misma información**, pero con nombres de atributo diferentes.
- Deberían llegar/salir del ViewModel hacia la UI como **una entidad nueva/común**.

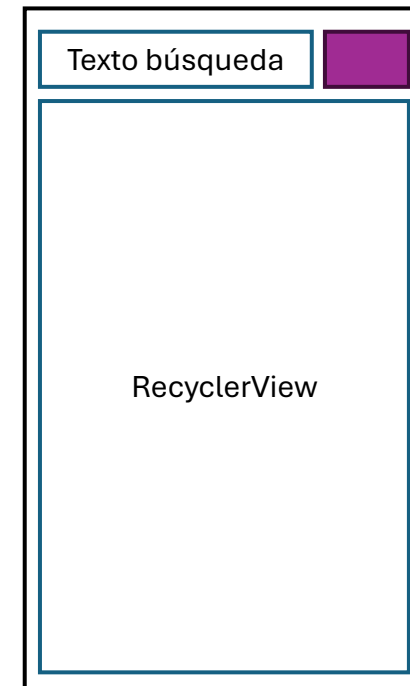


# Adapter – Uno es suficiente

- Modifica el código para trabajar con un único adapter.
- Coleccionará objetos de la nueva entidad: PeliculaUI

# Implementa la búsqueda

- Debe hacer uso del repositorio.
  - En el código de retrofit tienes un método para realizar la búsqueda.
- Debe hacer uso de un viewModel.
- Debe utilizar el mismo adapter.
- Al hacer clic en el botón  :
  - Se cargará en el RecyclerView los resultados.



# Caché - ¿Peticiones excesivas?

- ¿Cómo evitarías el uso de peticiones excesivas?
- ¿Qué código se vería afectado?