

Interactuando con la UI



KEEPCODING
Tech School



Acceso a las vistas

Acceso en Kotlin a las vistas

Es posible acceder a las vistas definidas en el XML de dos formas:

```
findViewById(...)
```

```
twContent.text = etContent.text
```

o a través de un binding:

```
private lateinit var binding : ActivityMainBinding
```

y

```
binding = ActivityMainBinding.inflate(layoutInflater)
```

```
setContentView(binding.root)
```



Gestión de las vistas

Accediendo/Modificando los atributos

Es posible acceder al contenido de una vista utilizando los respectivos métodos disponibles.

Estos métodos dependen del tipo de la vista. Por ejemplo,

```
twContent.text = etContent.text
```

siendo

`twContent` un `TextView` y `etContent` un `EditText`

Listeners

Un listener es una función que se queda en espera hasta que cierto evento ocurre, momento en el que se ejecuta. Ejemplos:

```
bConvert.setOnClickListener {...}
```

```
twContent.setOnLongClickListener {...}
```

```
etContent.doAfterTextChanged {...}
```

Proyecto Resumen

<https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/05d08649c8df2310e1f221278c51ed7b7d67f848>



GitHub

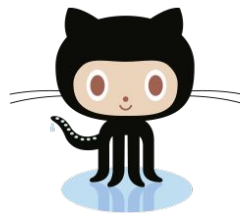
Ejercicio

Teniendo una Activity con 2 EditText, un TextView y un botón...

- El botón debe estar desactivado hasta que se hayan introducido algún carácter en los 2 EditText. Atributo "Enabled".
- Cuando el botón se pulsa, al texto del EditText1 se le añade el texto que hubiera en el EditText2. Posteriormente se borra el contenido de EditText2, en consecuencia el botón queda de nuevo desactivado.
- El TextView debe mostrar el TAG de la vista con el que el usuario se encuentre interactuando en ese momento. Propiedad Focus.

Proyecto Resumen

<https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/9fbd7842f04a4b9ff463e0eec94399b8a3d2670e>



GitHub

Ejercicio Destacado



- Haz que cuando se pulse el botón Login, si el checkBox está activado, se guarde en las SharedPreferences el usuario
- Haz que cuando se inicie la app, si hubiera un usuario guardado en las sharedPreferences, el editText debe mostrarse relleno con dicho usuario
- Si se desactiva el checkBox, se debe borrar lo que haya en las sharedPreferences.





| Clases en Android

Un proyecto Android puede contener tantas Clases como se desee.

La forma de crear nuevas clases y utilizarlas es similar a cualquier otro proyecto de otra plataforma.



Testing

| ¿Cómo se prueban las ViewModel?

Activities -> UnitTest

- Inconvenientes que conllevan:
 - No hay activity
 - No hay fragment
 - No hay Context

Ejemplo


Proyecto en GitHub



GitHub



StateFlow



| ¿Qué es?

StateFlow es una clase que contiene un elemento de tipo “observable”.

Notifica a “quien esté escuchando” de los cambios en la variable que contiene.

[Documentación](#)

¿Dónde emito?

Desde el **viewModel** creamos la variable `StateFlow`

```
private val uiState = MutableStateFlow(0)
```

```
val uiState: StateFlow<Int> = uiState
```

Al cual podemos cambiar el valor

```
uiState.value++
```

¿Dónde escucho?

Desde la **Activity** creamos la variable StateFlow

```
lifecycleScope.launch {
```

```
    viewModel.uiState.collect {
```

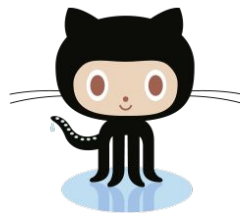
```
        println(it)
```

```
    }
```

```
}
```

Ejemplo

<https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/0aa2496c4e8719fd7a26f124066747603f1f4b52>



GitHub

Preparando el futuro

Si utilizamos un **sealed class** podemos enviar los datos que dispongamos de la App

```
sealed class UiState {
```

```
    data class Num(val number: Int): UiState()
```

```
    object Finished: UiState()
```

```
}
```



Ejercicio

Partiendo del ejercicio de la Activity con 2 EditText, un TextView y un botón...

- Haz uso de los StateFlow para llevar toda la lógica posible al ViewModel

Testing

| ¿Cómo se prueban el StateFlow?

```
viewModel.sumarUno()
```

```
assertEquals(viewModel.uiState.value, 1)
```

```
viewModel.sumarUno()
```

```
assertEquals(viewModel.uiState.value, 2)
```

Ejemplo

Proyecto en GitHub

(Visto durante la clase)

[https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/7aa859a3adbf
a19a6b381ea9aa749ce74abf64ed](https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/7aa859a3adbf
a19a6b381ea9aa749ce74abf64ed)



GitHub

(Solución posterior)

[https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/527c0f775985
5d244024c6aec8a8ef02bc851448](https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/527c0f775985
5d244024c6aec8a8ef02bc851448)



Adapters



Adapters, el gestor de listas

La forma más eficiente de mostrar una gran cantidad de elementos similares es utilizando un adapter. Serán necesarios 3 pasos:

- Definir el RecyclerView, donde ubicamos todos los elementos dentro de la UI
- Definir un ViewHolder, donde gestionaremos la UI de cada elemento
- Crear un Adapter, donde organizaremos al conjunto de ViewHolders

Proyecto resumen

Proyecto en GitHub



GitHub

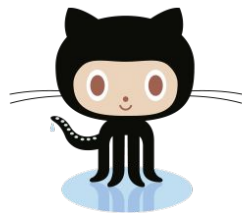
Ejercicio

Partiendo del ejercicio anterior realiza lo siguiente:

- Incrementar el número de elementos a 10.
- En las posiciones impares, el texto debe cambiar a: "Soy impar"
- Cambiar el color del background de los pares al color que quieras.
- Añadir un elemento al final que diga "Soy el último"

Solución

<https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/6e4602289d1c19568b2bb2d25637a9410d3d706b>



GitHub

Ejercicio

Partiendo del ejercicio realiza lo siguiente:

- Cambiar el nombre a “Soy el último” por “Añadir”
- Cuando se pulsa el último elemento, debe aparecer un Toast que diga: “Añadiendo”
- Cuando se pulsa el último elemento, se debe añadir un nuevo elemento a adapter

Adapters con lista I

La forma más eficiente de mostrar listas en Android es mediante la combinación de:

Adapter:

```
class StringAdapter(var stringList : List<String>) :  
RecyclerView.Adapter<StringAdapter.StringViewHolder>()
```

```
adapter = StringAdapter(values)
```

ViewHolder:

```
class StringViewHolder(root: View, var textView: TextView) : RecyclerView.ViewHolder(root)
```

Adapter con lista II

RecyclerView:

```
<androidx.recyclerview.widget.RecyclerView
```

```
    android:id="@+id/recyclerView"
```

```
    android:layout_width="match_parent"
```

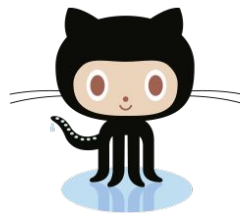
```
    android:layout_height="match_parent"/>
```

```
recyclerView.setLayoutManager(new LinearLayoutManager(this))
```

```
recyclerView.setAdapter(adapter)
```


Proyecto resumen

<https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/6e4602289d1c19568b2bb2d25637a9410d3d706b>



GitHub

Escuchando eventos de un adapter

Un comportamiento típico de las listas es que cuando se pulse sobre un elemento ocurran cosas.

Por tanto será necesario saber dónde se ha realizado la pulsación:

```
holder.root.setOnClickListener
```

Y obtener a qué elemento de la lista le corresponde:

```
val pos = recyclerView.getChildLayoutPosition(it)
```

Ejercicios

<https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/6e4602289d1c19568b2bb2d25637a9410d3d706b>



GitHub



Ejercicio

Desarrolla una app que muestre todos los personajes incluidos en el siguiente [archivo](#)

Los personajes deben aparecer en fila, poniendo su imagen, su nombre y su raza

Debes utilizar un RecyclerView, Adapter, y ViewHolder

Enviando información de la Activity a su adapter

Añadimos una función al adapter que permite recibir vía parámetro los datos que queremos pasarle al adapter.

Guardamos al adapter en una variable local

Enviamos la información llamando desde la activity a ese método

Enviando información del Adapter a su Activity I

- La comunicación entre la Activity y el Adapter es muy sencilla y ya hemos realizado varios ejemplos en los que ocurre. Simplemente se crea una función en el adapter que se llama desde la Activity.

Ej:

```
adapter.addPreguntaToList(pregunta)
```

- La comunicación en sentido contrario Adapter - Activity no es tan sencilla ya que el Adapter no sabe de la existencia de la activity. Por ello, debemos utilizar callbacks o listeners.

Enviando información del Adapter a su Activity II

Paso 1:

- Definir una interfaz e implementarla en la Activity. Ej:

```
interface MyListener {fun onSomeAction()}
```

```
class MainActivity : AppCompatActivity(), MyListener {
```

```
    override fun onSomeAction() {...}
```

```
}
```

Enviando información del Adapter a su Activity III

Paso 2:

- Modificar nuestro adapter para recibir un parámetro de tipo `MyListener`

```
class PreguntaAdapter(var listener : MyListener ) : RecyclerView.Adapter
```

- Pasar nuestra activity como parámetro al Adapter.

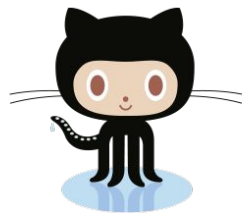
```
var adapter : PreguntaAdapter = PreguntaAdapter(this) // Siendo this, nuestra Activity que implementa MyListener
```

- Llamar al listener desde nuestro Adapter.

```
listener.doSomeAction()
```


Ejemplo resumen

<https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/6e4602289d1c19568b2bb2d25637a9410d3d706b>



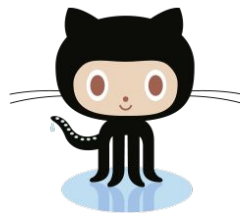
GitHub



Imágenes

Ejemplo resumen

<https://github.com/KeepCodingMobile16/Fundamentos-Android/commit/87ca90eefc6d2b0d7d520b8676918687364dc82d>



GitHub



Arquitectura



| ¿Dónde estamos?

Dentro de MVWM:

- Modelo
- Vista (Activity + Xml)
- **ViewModel**



| ¿Objetivo?

Realizar el trabajo duro

Sus tareas son:

- Nutrir de datos a la Activity
- Gestionar y realizar los cálculos que se solicite la Activity

Fin

