Reseña:

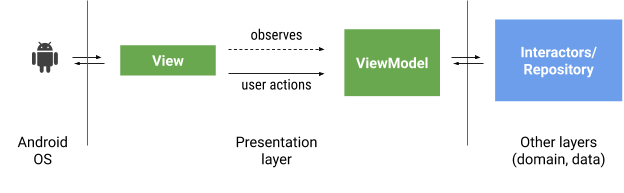
Viewbinding

Esta es la primera forma de vincular las vistas, es por decirlo de algún modo la forma original

Desde hace un tiempo, existe en el desarrollo Android una librería conocida como Data Binding.

Esta librería permite acceder a las vistas de una forma muy sencilla, enlazando variables de nuestro código Kotlin o Java con los componentes del XML.

Esto se consigue gracias a que habilita la posibilidad de añade código dentro de los XMLs.

Gráfico de Data Binding extraído de code-labs.io

Esta solución es muy interesante, pero quizá demasiado invasiva para algunos desarrolladores, que ven cómo sus XMLs se empiezan a llenar de código. Bien usado es muy interesante, pero es muy fácil usarlo mal.

Además, se necesitan modificar los XMLs donde se usan, y aumenta los tiempos de compilación de forma sensible.

Pero hay **un subconjunto de Data Binding que es muy sencillo de usar** y que simplifica mucho la vida.

Se trata de unos objetos llamados Binding, que contenen todas las vistas del XML casteados al tipo correcto, además teniendo en cuental a posible nulidad de cada vista.

Esto simplifica mucho el trabajo, ya que en una sola línea podemos recuperar todas las vistas y tenerlas preparadas para su uso.

Con View Binding simplemente han extraído esta funcionalidad, la han optimizado en tiempo de compilación, y nos han ahorrado la necesidad de modificar el XML para poder usarlo.

**En literalmente 2 líneas de código**, tendrás tus XML listos para ser convertidos en objetos de binding.

## Configuración de View Binding

La forma de configurarlo depende de la versión de Android Studio. Para Android Studio 4.0 y siguientes, en el build.gradle poner lo siguiente:

android {

buildFeatures {

viewBinding = **true**

}

}

Si es anterior, se usa lo siguiente:

android {

viewBinding.enabled = **true**

}

## Cómo usar View Binding en una Activity

Lo único que necesitas es modificar la forma en que inflas la vista. En vez de llamar a setContentView con el identificador del layout, lo harás pasándole la vista que has inflado previamente con View Binding, de esta forma:

**val** binding = ActivityMainBinding.inflate(layoutInflater)

setContentView(binding.root)

binding.root contiene la raíz del layout que ha sido inflado previamente. Ya solo necesitas acceder a las properties del objeto para usar sus vistas:

binding.button.setOnClickListener {

...

}

## Cómo usar View Binding en un Adapter de RecyclerView

Aquí hay al menos un par de formas de hacerlo:

* Usando el método inflate en el onCreateViewHolder y almacenando el objeto binding
* Inflando de forma clásica el objeto y usando el método bind en el ViewHolder para tenerlo allí disponible.

A mí personalmente me gusta más la segunda forma, porque no hay que andar preocupándose de pasar el objeto de un punto a otro del adapter, sino que se crea justo donde se necesita.

override **fun** onCreateViewHolder(parent: ViewGroup, viewType: **Int**): ViewHolder {

**val** v = parent.inflate(R.layout.view\_media\_item, **false**)

**return** ViewHolder(v, listener)

}

**class** ViewHolder(view: View, private **val** listener: MediaListener) :

RecyclerView.ViewHolder(view) {

private **val** binding = ViewMediaItemBinding.bind(view)

}

A partir de ese punto, ya se puede usar exactamente igual que lo hicimos en la Activity.

## Cómo usar View Binding en una Custom View

Aquí es muy similar a la Activity: inflamos la vista, nos guardamos el binding, y lo usamos donde necesitemos:

**class** MediaItemView *@JvmOverloads* **constructor**(

context: Context, attrs: AttributeSet? = **null**, defStyleAttr: **Int** = 0

) : FrameLayout(context, attrs, defStyleAttr) {

private **val** binding = ViewMediaItemBinding

.inflate(LayoutInflater.from(context), **this**, **true**)

**fun** setItem(mediaItem: MediaItem) {

with(binding) {

mediaThumb.loadUrl(mediaItem.url)

mediaTitle.text = mediaItem.title

}

}

}

## Cómo migrar de Synthetics de Kotlin Android Extensions a View Binding

Los Synthetics se están quedando un poco obsoletos, y tengo la impresión de que esto va a ser útil tarde o temprano, así que te voy a contar qué es lo que necestarías para migrar.

Lo más sencillo es que infles la vista con View Binding, que sustituyas el setContentView pasando la vista, y que hagas un apply sobre el objeto binding para que el resto del código se quede igual.

Un código como este:

setContentView(R.layout.activity\_main)

recycler.adapter = MediaAdapter(getItems())

Se transformaría en uno como este:

ActivityMainBinding.inflate(layoutInflater).apply {

setContentView(root)

recycler.adapter = MediaAdapter(getItems())

}

Como ves es muy sencillo. Incluso he creado un Live Templates para activities que te puede simplificar las cosas. Si quieres saber cómo importarlo, echa un vistazo a [mi artículo sobre Live Templates](https://devexperto.com/live-templates-android-studio-intellij/):

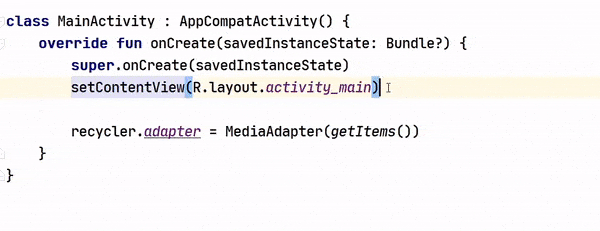
$VIEW\_BINDING\_CLASS$.inflate(layoutInflater).apply {

setContentView(root)

$SELECTION$$**END**$

}

Lo único que tienes que hacer es seleccionar el bloque antiguo de código, hacer Ctrl + Alt + J, y seleccionar el Live Template correspondiente:



## ¿Por qué debería usarlo frente a otras alternativas?

La realidad es que el resto de opciones aún siguen siendo válidas, pero con matices.

ButterKnife, la más popular, a día de hoy está deprecada en favor de esta nueva librería.

Por otro lado, los synthetics son un poco más “oscuros”, ya que no tenemos mucho control sobre cómo funcionan, y además siempre dejan en manos del programador la decisión sobre la nulidad de cada vista. No asisten de ninguna forma en cuanto a esto.

Esta tabla está extraída del [artículo en el blog de Android Developers](https://medium.com/androiddevelopers/use-view-binding-to-replace-findviewbyid-c83942471fc):

