# 2.5.1 RecyclerView

RecyclerView nos va a permitir mostrar en pantalla colecciones de datos. Se va a sustentar sobre otros componentes complementarios para determinar cómo acceder a los datos y cómo mostrarlos.

|  |  |
| --- | --- |
|  |  |

En concreto se apoya en un **adaptador** para trabajar con los datos. A su vez, dicho adaptador se apoya en un patrón **view holder** que se encarga de contener y gestionar las vistas asociadas a cada elemento individual de la lista.

El control RecyclerView se encargará de crear tantos view holder como sea necesario para mostrar los elementos de la lista que se ven en pantalla y los gestionará eficientemente de forma que no tenga que crear nuevos objetos para mostrar más elementos de la lista al hacer scroll, sino que tratará de «reciclar» aquellos que ya no sirven por estar asociados a otros elementos de la lista que ya han salido de la pantalla.

Vamos a verlo mejor con un ejemplo:

* Añadimos un nuevo RecyclerView al layout de nuestra actividad principal. Lo hacemos con el diseñador.

|  |
| --- |
| <androidx.recyclerview.widget.RecyclerView  android:id="@+id/vRecycler"  android:layout\_width="409dp"  android:layout\_height="wrap\_content" /> |

* Creamos la clase que encapsulará los datos que van a componer cada elemento a mostrar. En nuestro caso dos textos.

|  |
| --- |
| data class Titular(val titulo: String, val subtitulo: String) |

* En cada elemento de la lista queremos mostrar ambos datos, por lo que el siguiente paso será crear un layout XML con la **estructura visual que deseemos**. En mi caso voy a mostrarlos en dos etiquetas de texto (TextView), la primera de ellas en negrita y con un tamaño de letra un poco mayor. Llamaremos a este layout «listitem\_titular.xml» y lo colocaremos en la carpeta /app/res/layout. Lo confeccionamos con el diseñador también.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <TextView android:id="@+id/lblTitulo"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:textStyle="bold"  android:textSize="24sp" />  <TextView android:id="@+id/lblSubtitulo"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:textStyle="normal"  android:textSize="16sp" />  </LinearLayout> |

* A continuación escribiremos nuestro **adaptador**. Este adaptador deberá extender a la clase RecyclerView.Adapter, de la cual tendremos que sobrescribir principalmente tres métodos:
  + onCreateViewHolder(). Encargado de crear los nuevos objetos ViewHolder necesarios para los elementos de la colección.
  + onBindViewHolder(). Encargado de actualizar los datos de un ViewHolder ya existente.
  + onItemCount(). Indica el número de elementos de la colección de datos.

|  |
| --- |
| import android.view.LayoutInflater  import android.view.View  import android.view.ViewGroup  import android.widget.LinearLayout  import android.widget.TextView  import androidx.recyclerview.widget.RecyclerView  //Creamos el adaptador con las propiedades “datos” y un escuchador.  class AdaptadorTitulares(private val datos: Array<Titular>,  **private val clickListener : (Titular) -> Unit**) :  RecyclerView.Adapter<AdaptadorTitulares.TitularesViewHolder>() {  // El .Adapter necesita que le especifiquemos el view holder a usar <>.  /\* Creamos el view holder\*/  class TitularesViewHolder(val item: View) : RecyclerView.ViewHolder(item) {  val lblTitulo = item.findViewById(R.id.lblTitulo) as TextView  val lblSubtitulo = item.findViewById(R.id.lblSubtitulo) as TextView  fun bindTitular(titular: Titular){  lblTitulo.text = titular.titulo  lblSubtitulo.text = titular.subtitulo  }  }  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TitularesViewHolder {  val item = LayoutInflater.from(parent.context)  .inflate(R.layout.listitem\_titular, parent, false) as LinearLayout  return TitularesViewHolder(item)  }  override fun onBindViewHolder(holder: TitularesViewHolder, position: Int) {  val titular = datos[position]  holder.bindTitular(titular)  holder.item.setOnClickListener{**clickListener**(titular)};  }  override fun getItemCount() = datos.size  } |

En el método **onCreateViewHolder()** nos limitaremos a inflar (construir) una vista a partir del layout correspondiente a los elementos de la lista (listitem\_titular), y crear y devolver un nuevo ViewHolder llamando al constructor de nuestra clase TitularesViewHolder pasándole dicha vista como parámetro.

Los dos métodos restantes son aún más sencillos. En **onBindViewHolder()** tan sólo tendremos que recuperar el objeto Titular correspondiente a la posición recibida como parámetro y asignar sus datos sobre el ViewHolder también recibido como parámetro. Por su parte, **getItemCount()** tan sólo devolverá el tamaño del array de datos.

* Ya solamente nos quedaría cargar de información nuestro RecyclerView.

|  |
| --- |
| //Código para cargar el ReyclerView  var recView:RecyclerView = findViewById(R.id.vRecycler)  val datosRec =  Array(50) { i -> Titular("Titulo $i", "Subtítulo Item $i") }  val adaptadorRec = AdaptadorTitulares(datosRec){  Log.i("DemoRecView", "Pulsado el elemento: ${it.titulo}")  }  /\*  {…} Es una **lambda** que representa el clickListener que configuraste en la implementación de AdaptadorTitulares.  Esta lambda se ejecutará cada vez que un usuario haga clic en un elemento del RecyclerView.  Dentro de la lambda, it se refiere al elemento de tipo Titular que fue clicado.  \*/  recView.setHasFixedSize(true)  recView.layoutManager =  //LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false)  GridLayoutManager(this, 3)  recView.adapter = adaptadorRec |

## 2.5.1.1 Responder a eventos

La clase RecyclerView no incluye un evento onItemClick. Se delega esta tarea a otro componente, en este caso a la propia vista que conforma cada elemento de la colección.

En nuestro caso aprovecharemos el método onBindViewHolder() para asignar a su vista asociada el evento *onClick*.

Para poder asignar el evento desde fuera del adaptador, incluiremos la función a llamar como nuevo atributo del adaptador. Adicionalmente, por comodidad, pasaremos como parámetro de esta función el dato (en nuestro ejemplo el objeto Titular) contenido en ese elemento. (ver código fuente)

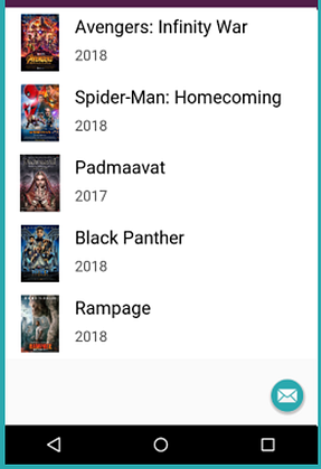
Por su parte, para añadir la implementación del evento desde la actividad principal, modificaríamos la creación del adaptador de la siguiente forma:

|  |
| --- |
| //...  val datos =  Array(50) { i -> Titular("Titulo $i", "Subtítulo Item $i") }  val adaptador = AdaptadorTitulares(datos) {  Log.i("DemoRecView", "Pulsado el elemento: ${it.titulo}")  }  //... |

En este caso al pulsar sobre un elemento de la lista tan sólo escribiremos, por ejemplo, un mensaje de log indicando la posición en la lista del elemento pulsado. Podéis ver los mensajes de log de la aplicación en las vistas inferiores llamadas Run y Logcat de Android Studio.

# 2.5.2 Actividad 1

Crea una aplicación con un solo Activity que luzca como se aprecia en la siguiente imagen. Hacer uso del widget RecyclerView.



# 2.5.3 Webgrafía

<https://www.sgoliver.net/blog/controles-de-seleccion-v-recyclerview/>

<https://www.develou.com/recyclerview-en-android/>