



# Guía de uso “RecyclerView”

Walter Martín Lopes



## ¿Qué es un RecyclerView?

Un *RecyclerView* al igual que un *ListView* es un componente utilizado para mostrar listas de elementos.

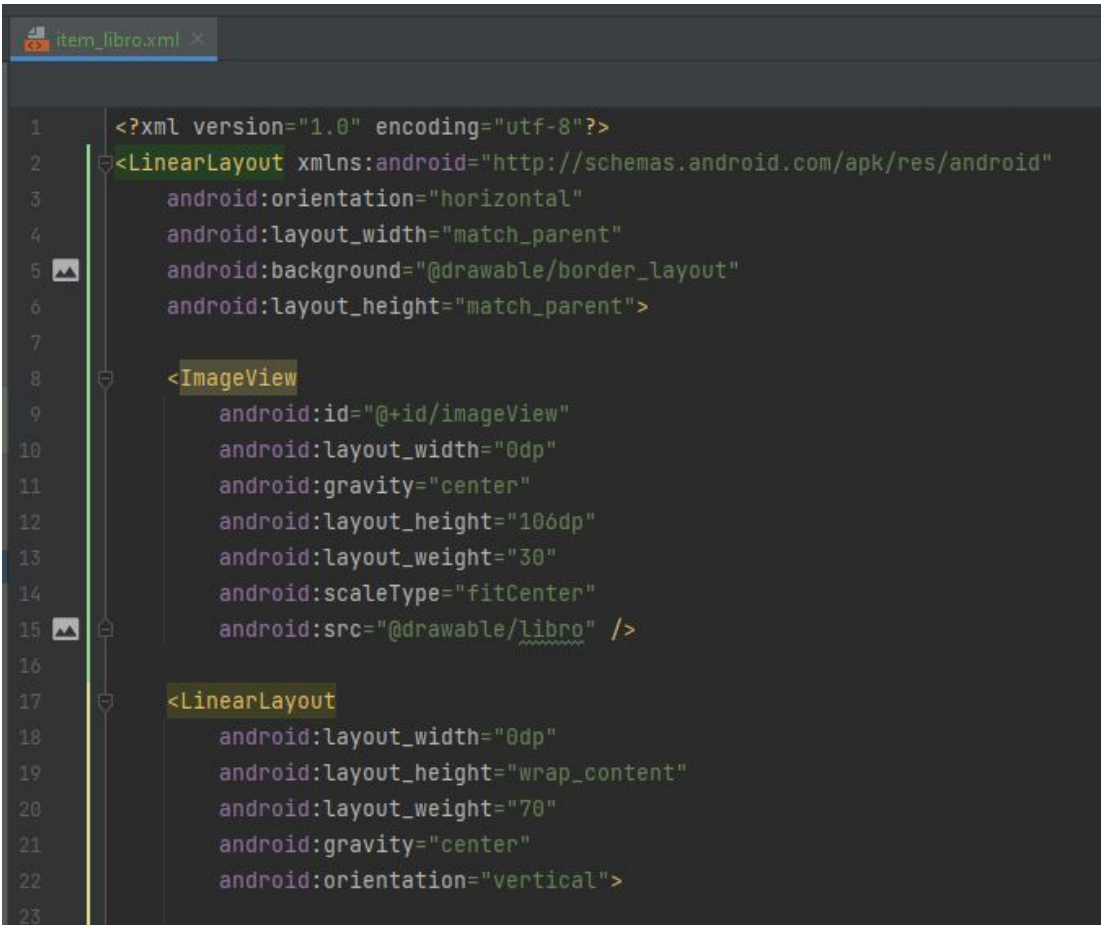
*RecyclerView* es más moderno y versátil que *ListView*. Ofrece mejor rendimiento, especialmente en listas largas o complejas, ya que **reutiliza** las vistas de los elementos a medida que se desplazan fuera de la pantalla (*de ahí su nombre, recicla las vistas*). Esto lo hace más eficiente en términos de memoria y procesamiento.

Además, *RecyclerView* es más flexible ofreciendo más opciones para personalizar la forma en que se muestran los elementos (como en una cuadrícula, por ejemplo), y es más fácil agregar animaciones o diferentes tipos de vistas dentro de la lista.

## Guía de uso

Explicaremos su funcionamiento usando de ejemplo la práctica entregable del tema 9, mostrando paso a paso cómo hacer uso de este componente con una vista personalizada.

En primer lugar debemos crear un nuevo archivo de layout personalizado **XML** en la carpeta '*res/layout*'. Al igual que como hacíamos con los **ListView**, este será el diseño para cada elemento de la lista.



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="horizontal"
4      android:layout_width="match_parent"
5      android:background="@drawable/border_layout"
6      android:layout_height="match_parent">
7
8      <ImageView
9          android:id="@+id/imageView"
10         android:layout_width="0dp"
11         android:gravity="center"
12         android:layout_height="106dp"
13         android:layout_weight="30"
14         android:scaleType="fitCenter"
15         android:src="@drawable/libro" />
16
17     <LinearLayout
18         android:layout_width="0dp"
19         android:layout_height="wrap_content"
20         android:layout_weight="70"
21         android:gravity="center"
22         android:orientation="vertical">
23
```

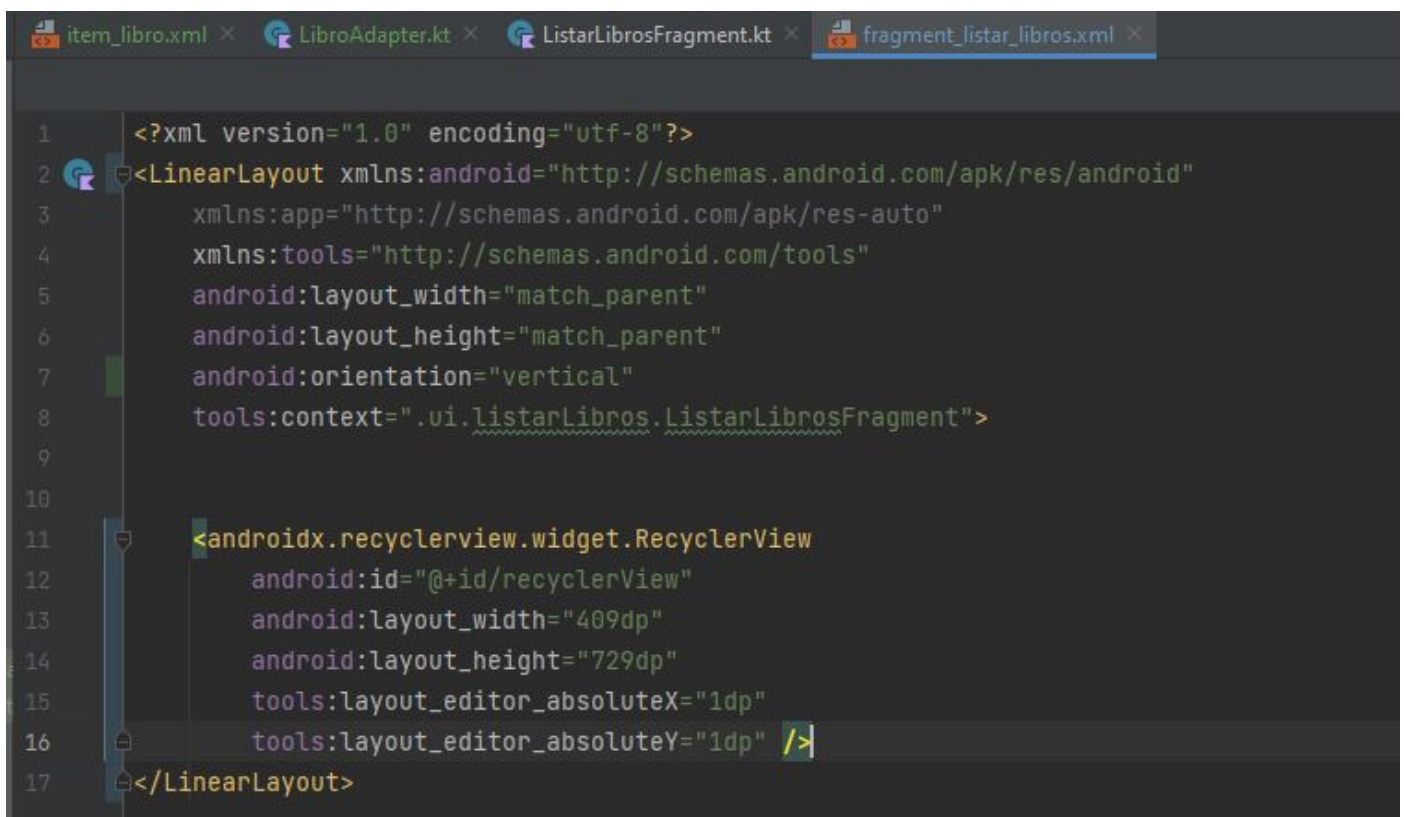
## Guía de uso

Seguidamente crearemos un adaptador para el *RecyclerView*, parecido a como hacíamos con los *ListView*. Este adaptador será el que vincula los datos con los *Views* en el layout del ítem. La clase debe extender de '*RecyclerView.Adapter*'.

```
item_libro.xml x LibroAdapter.kt x
12
13 new *
14 class LibroAdapter(private val listaLibros: List<Libro>) :
15     RecyclerView.Adapter<LibroAdapter.LibroViewHolder>() {
16
17     new *
18     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): LibroViewHolder {
19         val itemView =
20             LayoutInflater.from(parent.context).inflate(R.layout.item_libro, parent, attachToRoot: false)
21         return LibroViewHolder(itemView)
22     }
23
24     new *
25     override fun onBindViewHolder(holder: LibroViewHolder, position: Int) {
26         //Recupero el libro de la posición actual
27         val libroActual = listaLibros[position]
28         // Configura los Views del holder
29         holder.textViewTitulo.text = libroActual.titulo
30         holder.textViewAutor.text = libroActual.titulo
31         holder.textViewPrecio.text = libroActual.titulo
32         //Utilizo la libreria Picasso para mostrar la imagen por la URL
33         Picasso.get()
34             .load(libroActual.imagen) //Cargamos la url de la imagen del campo imagen de nuestro objeto Libro
35             .placeholder(R.drawable.libro) // Imagen de placeholder
36             .error(R.drawable.libro) // Imagen de error
37             .into(holder.imageViewLibro) //Asigno el imageView
38     }
39
40     new *
41     override fun getItemCount() = listaLibros.size
42
43     new *
44     class LibroViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
45         //Defino los Views
46         val imageViewLibro: ImageView = itemView.findViewById(R.id.imageViewLibro)
47         val textViewTitulo: TextView = itemView.findViewById(R.id.textViewTitulo)
48         val textViewAutor: TextView = itemView.findViewById(R.id.textViewAutor)
49         val textViewPrecio: TextView = itemView.findViewById(R.id.textViewPrecio)
50     }
51 }
```

## Guía de uso

El siguiente paso será agregar el *RecyclerView* a nuestro layout del *Activity* o *Fragment* como en nuestro caso. Se agrega de igual manera que cualquier otro componente.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical"
8     tools:context=".ui.listarLibros.ListarLibrosFragment">
9
10
11     <androidx.recyclerview.widget.RecyclerView
12         android:id="@+id/recyclerView"
13         android:layout_width="409dp"
14         android:layout_height="729dp"
15         tools:layout_editor_absoluteX="1dp"
16         tools:layout_editor_absoluteY="1dp" />
17 </LinearLayout>
```

## Guía de uso

Por último debemos configurar el **RecyclerView** en nuestro **Fragment** vinculando el adaptador con el **RecyclerView**, usaremos un '**LinearLayoutManager**' para mostrar los ítems en forma de lista vertical.

```
1  ListarLibrosFragment.kt
2
3  import ...
4
5  21
6
7  * vjp-walterML *
8
9  class ListarLibrosFragment : Fragment() {
10
11      //VARIABLES GLOBALES
12      private var _binding: FragmentListarLibrosBinding? = null
13
14      // This property is only valid between onCreateView and onDestroyView.
15      * vjp-walterML
16      private val binding get() = _binding!!
17      private lateinit var recyclerView: RecyclerView
18
19      * vjp-walterML *
20
21      override fun onCreateView(
22          inflater: LayoutInflater,
23          container: ViewGroup?,
24          savedInstanceState: Bundle?
25      ): View {
26          _binding = FragmentListarLibrosBinding.inflate(inflater, container, attachToParent: false)
27          val root: View = binding.root
28
29          //LÓGICA=====
30          //Recupero el recycledView
31          recyclerView = root.findViewById(R.id.recyclerView)
32          //Vinculo el adaptador con el RecyclerView y uso un LinearLayoutManager para mostrar los ítems en forma de lista vertical.
33          recyclerView.layoutManager = LinearLayoutManager(requireContext())
34          //Lanzo petición
35          lanzarPetición()
36
37          return root
38      }
39
40      * vjp-walterML
41      override fun onDestroyView() {
42          super.onDestroyView()
43          _binding = null
44      }
45
46      ..
```

# Guía de uso

```
ListarLibrosFragment.kt
54
55 //=====
56 new "
57 private fun lanzarPetición() {
58     Log.d( tag: "DebugLog", msg: "lanzarPetición iniciada")
59     // Lanza la corrutina en segundo plano
60     CoroutineScope(Dispatchers.IO).launch { this: CoroutineScope
61         try {
62             Log.d( tag: "DebugLog", msg: "Dentro de la corrutina")
63             // Petición HTTP para obtener la lista de todos los libros
64             val moduleResponse: Response<List<Libro>> =
65                 LibroRetrofit.apiService.getAllLibros()
66             Log.d( tag: "DebugLog", msg: "Petición API realizada")
67             var libros: List<Libro>?
68             if (moduleResponse.isSuccessful) {
69                 Log.d( tag: "DebugLog", msg: "Respuesta exitosa")
70                 // Se obtiene la lista de objetos modulo del body del objeto Response
71                 libros = moduleResponse.body()
72
73                 if (libros != null) {
74                     Log.d( tag: "DebugLog", msg: "Lista de jugadores no es nula, tamaño: `${libros.size}`")
75                     mostrarModulosRecyclerView(libros)
76                 } else {
77                     Log.d( tag: "DebugLog", msg: "Lista de jugadores es nula")
78                 }
79             } else {
80                 Log.e( tag: "ErrorLog", msg: "Respuesta fallida, código de error: `${moduleResponse.code()}`")
81                 when (moduleResponse.code()) {
82                     404 -> showPlayerNotFoundError()
83                     500 -> showInternalServerError()
84                     else -> showGeneralError()
85                 }
86             }
87         } catch (e: Exception) {
88             Log.e( tag: "ErrorLog", msg: "Excepción en lanzarPetición: `${e.message}`")
89         }
90     }
91     }
92 new "
```



## Guía de uso

```
new *
private fun mostrarModulosRecyclerView(listaLibros: List<Libro>) {
    GlobalScope.launch(Dispatchers.Main) { this: CoroutineScope
        //Creo adaptador para el RecyclerView
        val adapter = LibroAdapter(listaLibros)
        //Le asigno el adapter
        recyclerView.adapter = adapter
    }
}
```

Con esto habríamos terminado de configurar nuestro **RecyclerView**.

También permite manejar eventos de *click* en cada ítem del **RecyclerView**. Una forma común de hacerlo es proporcionar una interfaz de callback en tu adaptador y configurar los listeners en **onBindViewHolder**.



# Guía de uso

Como resultado final, en esta imagen observamos el funcionamiento del *RecyclerView*.

