

## RECYCLERVIEWS: Explicación y ejemplo sencillo.

En esta app, aprenderemos qué son y cómo se implementan uno de los elementos más utilizados en aplicaciones móviles. Las listas, en concreto con RecyclerViews.

Un **RecyclerView** en Android es un componente versátil y eficiente para mostrar listas grandes de datos de manera eficiente. A diferencia de su predecesor, **ListView**, el **RecyclerView** está diseñado para optimizar el rendimiento al "reciclar" las vistas que ya no son visibles en pantalla, en lugar de crear nuevas vistas para cada elemento, lo que reduce el uso de memoria y mejora el rendimiento.

### Características principales:

- **Recycler**: Reutiliza las vistas que ya no son visibles en la pantalla para reducir la creación de nuevas vistas.
- **LayoutManager**: Permite personalizar cómo se organiza el contenido (por ejemplo, en una lista vertical, en una cuadrícula, etc.).
- **Adapter**: Actúa como puente entre los datos y las vistas, vinculando los datos a las vistas del **RecyclerView**.
- **ViewHolder**: Optimiza el acceso a los widgets de la interfaz de cada ítem al almacenar las vistas de un solo elemento.

### Flujo general:

1. Creas un **ViewHolder** y un **Adapter** que especifica cómo se debe vincular cada dato con una vista.
2. Se utiliza un **LayoutManager** para decidir cómo organizar las vistas en la pantalla.
3. El **RecyclerView** recicla y reutiliza vistas antiguas, para no tener que crear nuevas vistas cada vez que se desplaza por la lista.

### Ejemplo: Lista de Cantantes con interacción

Vamos a crear una lista de cantantes y al pulsar sobre un elemento, cambiará de color, y al dejarlo pulsado, lo eliminaremos de la lista.

#### 1. Paso 1: Estructura básica del RecyclerView

En el archivo XML (layout), añadimos el **RecyclerView** dentro de un **LinearLayout** (o cualquier otro contenedor):

*activity\_main.xml*

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

Añadimos un layout (item\_cantante.xml), que servirá para dar aspecto a cada ítem de mi RecyclerView.  
New→XML→LayoutXML File dentro de la carpeta layout.

Este layout simplemente tiene un TextView, en otros ejemplos le pondremos otro diseño.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="20dp">

    <TextView
        android:id="@+id/txCantante"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:textStyle="bold" />
</LinearLayout>
```

## 2. Paso 2: Crear el Adapter y el ViewHolder

### VIEWHOLDER

```
class CantanteViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val textView: TextView = itemView.findViewById(R.id.txCantante)
}
```

### ADAPTER

```
class AdaptadorCantantes(private val cantantes: ArrayList<String>) :
    RecyclerView.Adapter<CantanteViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
        CantanteViewHolder {
        // Inflamos el layout de cada elemento

        val view =
            LayoutInflater.from(parent.context).inflate(R.layout.item_cantante, parent,
                false)
        return CantanteViewHolder(view)
    }

    override fun onBindViewHolder(holder: CantanteViewHolder, position:
        Int) {
        // Inicializamos la lista de cantantes

        val cantante = cantantes[position]
        holder.textView.text = cantante

        // Al hacer clic, el fondo del ítem cambia de color
        holder.itemView.setOnClickListener {
```

```

        holder.itemView.setBackgroundColor(Color.LTGRAY)
    }

    // Al dejar pulsado, se elimina el ítem de la lista
    holder.itemView.setOnLongClickListener {
        cantantes.removeAt(position)
        notifyItemRemoved(position)
        notifyItemRangeChanged(position, cantantes.size)
        true
    }
}

override fun getItemCount(): Int {
    return cantantes.size
}
}

```

### 3. Paso 3: Configurar el RecyclerView en la actividad principal

```

class MainActivity : AppCompatActivity() {
    private lateinit var recyclerView: RecyclerView
    private lateinit var adapter: AdaptadorCantantes
    private lateinit var cantantes: ArrayList<String>

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }

        // Inicializamos la lista de cantantes
        cantantes = arrayListOf(
            "Billie Eilish",
            "Dua Lipa",
            "Harry Styles",
            "Olivia Rodrigo",
            "The Weeknd",
            "Doja Cat",
            "Bad Bunny",
            "Taylor Swift",
            "Rosalía",
            "Ariana Grande"
        )

        // Configuramos el RecyclerView
        recyclerView = findViewById(R.id.rvCantantes)
        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = AdaptadorCantantes(cantantes)
    }
}

```

## 4. Descripción del funcionamiento:

1. **Lista de cantantes actuales:** En **MainActivity**, hemos añadido una lista de cantantes populares actuales, como **Billie Eilish**, **Dua Lipa**, **The Weeknd**, **Bad Bunny**, etc.
2. **Adapter y ViewHolder:** El **CantanteAdapter** enlaza cada cantante de la lista a un **TextView** utilizando el layout simple android.R.layout.simple\_list\_item\_1.
3. **Interacciones:**
  - **Clic:** Al hacer clic sobre un ítem de la lista, el fondo del ítem cambia a color gris claro (**Color.LTGRAY**).
  - **Mantener pulsado:** Al mantener pulsado un ítem, este es **eliminado** de la lista, y el **RecyclerView** se actualiza automáticamente.

## 5. Resultado esperado:

Al ejecutar esta aplicación, verás una lista de cantantes actuales. Al hacer clic sobre un cantante, su ítem cambiará de color, y si mantienes pulsado, el cantante desaparecerá de la lista.

## 6. Opcional: Personalización del color

Si quieres personalizar el color que se aplica al hacer clic, puedes reemplazar **Color.LTGRAY** con cualquier otro color. Por ejemplo, podrías definir un color en tu archivo **res/values/colors.xml**:

## A TENER EN CUENTA

En un **RecyclerView**, los métodos **onCreateViewHolder()**, **onBindViewHolder()**, y el bloque **init** (o el constructor del adaptador) son los más importantes y fundamentales para gestionar cómo se muestran los elementos de la lista. Sin embargo, hay otros métodos y eventos relevantes en el ciclo de vida de un **RecyclerView** y su adaptador que te permiten un mayor control sobre el comportamiento y rendimiento. Aquí te menciono otros eventos importantes que puedes encontrar útiles:

## 7. Métodos clave adicionales en el ciclo de vida del RecyclerView:

1. **getItemCount()**
  - **Descripción:** Este método simplemente devuelve la cantidad de elementos que tiene el adaptador. El **RecyclerView** necesita esta información para saber cuántos elementos mostrar.
  - **Uso adecuado:** Aquí es donde informas al **RecyclerView** sobre el tamaño de la lista o conjunto de datos.

### Ejemplo:

```
override fun getItemCount(): Int {  
    return listItems.size  
}
```

1. **onViewRecycled()**
  - **Descripción:** Este método es llamado cuando una vista es reciclada (ya no es visible y se coloca en la "pila de reciclaje"). Esto te permite limpiar o restablecer cualquier recurso pesado que se haya asociado con la vista (como listeners o cargas pesadas).
  - **Uso adecuado:** Desasociar listeners, detener animaciones o limpiar imágenes en memoria.
  - **Ejemplo:**

kotlin

Copiar código

```
override fun onViewRecycled(holder: MiViewHolder) {
```

```

        super.onViewRecycled(holder)
        // Limpiar recursos si es necesario
    }

```

## 2. onViewAttachedToWindow()

- **Descripción:** Este método se llama cuando una vista es **adjuntada** a la ventana. Es útil para inicializar algún comportamiento o interacción específica cuando la vista vuelve a estar visible en la pantalla.
- **Uso adecuado:** Puedes usarlo para reiniciar alguna animación o restablecer un estado especial de la vista.
- **Ejemplo:**

```

kotlin
Copiar código
override fun onViewAttachedToWindow(holder: MiViewHolder) {
    super.onViewAttachedToWindow(holder)
    // Ejecutar alguna acción cuando la vista aparece en la pantalla
}

```

## 3. onViewDetachedFromWindow()

- **Descripción:** Se llama cuando una vista se **desasocia de la ventana**, lo que significa que ya no está visible. Se puede usar para detener cualquier proceso o animación que no deba seguir ejecutándose cuando la vista está fuera de pantalla.
- **Uso adecuado:** Pausar animaciones, guardar estados temporales.
- **Ejemplo:**

```

kotlin
Copiar código
override fun onViewDetachedFromWindow(holder: MiViewHolder) {
    super.onViewDetachedFromWindow(holder)
    // Detener animaciones o tareas que no deben seguir corriendo
}

```

# 8. Métodos de notificación de cambios en el RecyclerView:

## 5. notifyDataSetChanged()

- **Descripción:** Le indica al RecyclerView que **todos los datos han cambiado**, por lo que debe volver a renderizar toda la lista. Sin embargo, es poco eficiente y no se recomienda para grandes cambios, ya que vuelve a renderizar todos los elementos.
- **Uso adecuado:** Utilízalo con moderación, solo cuando todos los datos han cambiado significativamente.
- **Ejemplo:**

```

adapter.notifyDataSetChanged()

```

## 6. Otros métodos para notificar cambios del RecyclerView actualización:

- **notifyItemChanged(int position):** Notifica que un solo elemento ha cambiado.
- **notifyItemInserted(int position):** Notifica que un nuevo elemento ha sido insertado en una posición específica.
- **notifyItemRemoved(int position):** Notifica que un elemento ha sido eliminado.
- **notifyItemRangeChanged(int positionStart, int itemCount):** Notifica que un rango de elementos ha cambiado.
- **Ejemplo:**

```

adapter.notifyItemChanged(position)

```