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.

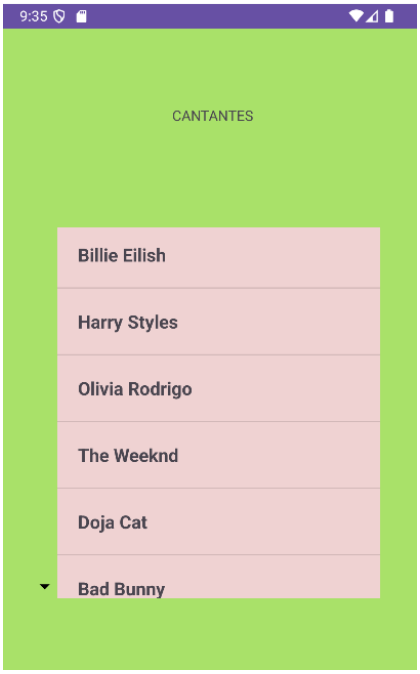
Características principales:

- **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.

Descripción del funcionamiento:

1. **Lista de cantantes actuales**: En CantanteProvider, 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.

Crea un proyecto: File→New Project → Empty Views Activity, lo llamaremos, por ejemplo, **RecyclerViewEjemploBasico**.

1. Paso 1: Estructura básica del RecyclerView

En el layout (activity_main.xml), añadimos el RecyclerView .

Después, añadimos un layout (item_cantante.xml), que servirá para dar aspecto a cada ítem de mi RecyclerView. Dentro de la carpeta layout (New→XML→LayoutXML)

item_cantante.xml 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 ViewHolder y el Adapter

VIEWHOLDER

En el viewHolder se suelen crear variables asociadas a las Views y en este caso, he añadido una variable tipo "bandera" que me indicará posteriormente en cada ítem, si está con el color inicial, para alternar el color cada vez que cambio.

```
class CantanteViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val binding = ItemCantanteBinding.bind(itemView)

    val textView: TextView = binding.txCantante
    var isBackgroundColorChanged = false // Flag to track color state
}
```

ADAPTER

En el adaptador además de vincular los datos con la Recycler, he creado algunos eventos para cada ítem. Por ejemplo, al hacer clic, cambia de color ese ítem, y si dejas pulsado, eliminas el elemento.

```
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 layoutInflater = LayoutInflater.from(parent.context)
        return CantanteViewHolder(layoutInflater.inflate(R.layout.item_cantante,
parent, false))
    }
    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 {
            if (holder.isBackgroundColorChanged) {
                holder.itemView.setBackgroundColor(Color.TRANSPARENT) // Or your
default color
                holder.isBackgroundColorChanged = false

            } else {
                holder.itemView.setBackgroundColor(Color.LTGRAY)
                holder.isBackgroundColorChanged = true
            }
        }

        // 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

Crear clase CantanteProvider donde tendremos guardada la lista de cantantes:

```
object CantanteProvider {
    val cantantesList = arrayListOf(
        "Billie Eilish",
        "Dua Lipa",
        "Harry Styles",
        "Olivia Rodrigo",
        "The Weeknd",
        "Doja Cat",
        "Bad Bunny",
        "Taylor Swift",
        "Rosalía",
        "Ariana Grande"
    )
}
```

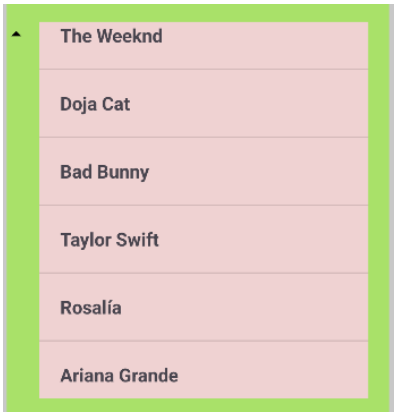
Iniciar RecyclerView en la actividad principal

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Configuramos el RecyclerView
        initRecyclerView()
    }

    private fun initRecyclerView() {
        val manager = LinearLayoutManager(this)
        binding.rvCantantes.layoutManager = manager
        binding.rvCantantes.adapter = AdaptadorCantantes(CantanteProvider.cantantesList)

        // Crear un DividerItemDecoration y agregarlo al RecyclerView
        val decoration = DividerItemDecoration(this, manager.orientation)
        binding.rvCantantes.addItemDecoration(decoration)

        // Agregar un ScrollListener al RecyclerView para mostrar/ocultar las flechas
        binding.rvCantantes.addOnScrollListener(object : RecyclerView.OnScrollListener() {
            override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
                val layoutManager = recyclerView.layoutManager as LinearLayoutManager
                val totalItemCount = layoutManager.itemCount
                val visibleItemCount = layoutManager.childCount
                val firstVisibleItemPosition = layoutManager.findFirstVisibleItemPosition()

                // Mostrar la flecha hacia abajo si no está en el final
                if ((firstVisibleItemPosition + visibleItemCount) < totalItemCount) {
                    // Mostrar flecha abajo
                    binding.imDown.visibility = View.VISIBLE
                } else {
                    // Ocultar flecha abajo
                    binding.imDown.visibility = View.GONE
                }

                // Mostrar la flecha hacia arriba si no está en el inicio
                if (firstVisibleItemPosition > 0) {
                    // Mostrar flecha arriba
                    binding.imUp.visibility = View.VISIBLE
                } else {
                    // Ocultar flecha arriba
                    binding.imUp.visibility = View.GONE
                }
            }
        })
    }
}
```

4. Mejoras realizadas

A) He añadido unas flechitas para que indique si mi RecyclerView tiene más elementos por arriba o por abajo. Esas flechitas las añades en la posición que quieras dentro de un imageview cada una, y luego este es el código que muestra su visibilidad o no, en el MainActivity, o dónde manipules en código tu RecyclerView. (en el ejemplo se activa solamente la de arriba, indicando que hay más elementos arriba)

B) Además, he añadido un separador entre ítem e ítem.



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:

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 listaItems.size
     }
     ```

2. **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**:

     ```
     override fun onViewRecycled(holder: MiViewHolder) {
        super.onViewRecycled(holder)
        // Limpiar recursos si es necesario
     }
     ```

3. **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**:

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

4. **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**:

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

Métodos de notificación de cambios en el RecyclerView:

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()

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)