Controles de selección (1)

Permiten elegir un valor entre un conjunto de posibles valores, facilitando la entrada de datos ya que hace innecesarias las comprobaciones de las entradas por teclado.

Existen varios controles de selección: *Spinner*, *ListView*, *RecyclerView*, *CardView*.

En todos ellos hemos de utilizar un adaptador (*Adapter*).

105

Controles de selección (2)

- **Adaptador: *Adapter***

Un adaptador es un objeto que actúa como interfaz entre el control y sus datos.

El adaptador, además de proporcionar los datos al control, se encargará de generar las vistas específicas de esos datos dentro del control de selección.

Los datos que se van a utilizar se pueden obtener de un *array*, un *string-array* (XML), una base de datos, etc.

106

Controles de selección (3)

- **Adaptador: *Adapter***

Android proporciona 3 tipos de adaptadores:

- *ArrayAdapter*: es el más sencillo, provee datos a un control de selección a partir de un array de objetos de cualquier tipo.
- *SimpleAdapter*: se utiliza para mapear datos sobre los diferentes controles definidos en un fichero XML de *layout*.

107

Controles de selección (4)

- **Adaptador: *Adapter***

- *SimpleCursorAdapter*: se utiliza para mapear las columnas de un cursor abierto en una base de datos sobre los diferentes elementos visuales contenidos en el control de selección.
- Adaptadores Personalizados: podemos crear nuevas clases a partir de cualquier *Adapter* para cumplir con nuestras necesidades.

108

Controles de selección (5)

- **Adaptador: *Adapter***

Al crear un adaptador hemos de establecer tres parámetros:

- Contexto: normalmente una referencia a la actividad desde la que se crea.
- ID *layout*: donde se mostrará cada uno de los datos. Puede ser uno predefinido de sistema o uno definido por nosotros.
- Datos: puede ser un *array* (Java), un recurso de tipo *string-array* (XML), etc.

109

Controles de selección (6)

- **Adaptador: *Adapter***
- Utilizando un *array*:

```
String[] datos = new String[]{"Elem1","Elem2","Elem3"};
ArrayAdapter<String> adaptador =new ArrayAdapter<String>
  (this, android.R.layout.simple_spinner_item, datos);
```

Se ha utilizado un *simple_spinner_item*, que es un *layout* predefinido de sistema que contiene únicamente un *TextView*, pero se podría haber utilizado otro *layout* definido por nosotros.

110

Controles de selección (7)

- **Adaptador: *Adapter***

- Utilizando un *string-array*:

```
ArrayAdapter<CharSequence> adaptador =
  ArrayAdapter.createFromResource (this, R.array.val_array,
  android.R.layout.simple_spinner_item);
```

valores_array.xml (dentro de values):

XML

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="val_array">
        <item>Elem1</item>
        <item>Elem2</item>
        <item>Elem3</item>
    </string-array>
</resources>
```

111

Controles de selección (8)

- **Lista desplegable: *Spinner***

Las listas desplegables en Android se denominan *Spinner*.

112

Controles de selección (9)

- **Lista desplegable: *Spinner***

XML

```
<Spinner
    android:id="@+id/spinnerOpciones"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

```
ArrayAdapter<CharSequence> adaptador =
  ArrayAdapter.createFromResource (this, R.array.val_array,
  android.R.layout.simple_spinner_item);
Spinner op = (Spinner)findViewById(R.id.spinnerOpciones);
adaptador.setDropDownViewResource
  (android.R.layout.simple_spinner_dropdown_item);
op.setAdapter(adaptador);
```

(*)*simple_spinner_dropdown_item* se utiliza para mostrar la lista emergente.

113

Controles de selección (10)

- **Lista desplegable: *Spinner***

En cuanto al tratamiento, el Listener más habitual es *setOnItemSelectedListener*, donde implementaremos dos métodos:

- *onItemSelected*
- *onNothingSelected*

```
op.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
  public void onItemSelected(AdapterView<?> av, View v, int pos, long id){
     TextView mensa = (TextView)findViewById(R.id.textMensa);
     mensa.setText("Seleccionado: " + av.getItemAtPosition(pos)); }
  public void onNothingSelected(AdapterView<?> av) {
     TextView mensa = (TextView)findViewById(R.id.textMensa);
     mensa.setText(""); }
});
```

114

Controles de selección (11)

- **Lista: *ListView***

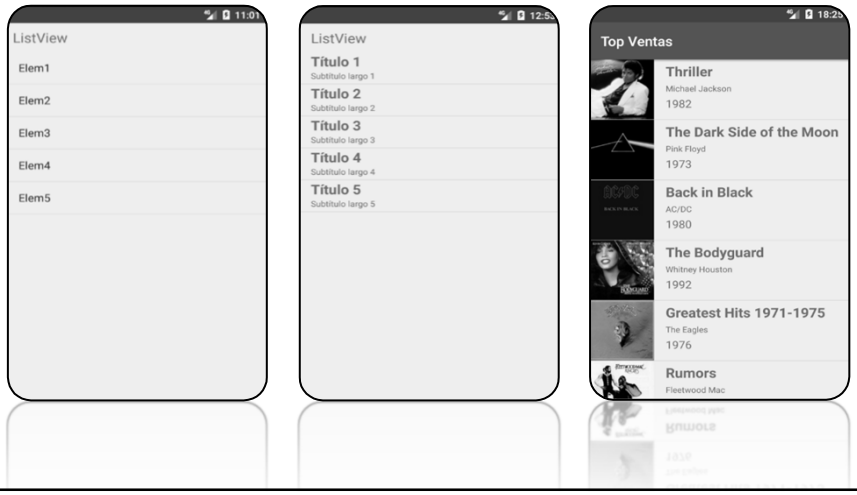
A diferencia del *Spinner,* los elementos de un *ListView* se muestran directamente sobre el control, de manera que si el número de elementos es mayor de las que se pueden mostrar habrá que añadir un *Scroll*.

Durante mucho tiempo fue el control de selección más utilizado, pero ha ido siendo sustituido por otros controles de selección más versátiles.

115

Controles de selección (12)

- **Lista: *ListView***



116

Controles de selección (11)

- **Lista: *ListView***

El proceso para crear una lista simple es casi idéntico al de un Spinner.

XML

```
<ListView
    android:id="@+id/listElem"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

```
ArrayAdapter<CharSequence> adaptador =
  ArrayAdapter.createFromResource (this, R.array.val_array,
  android.R.layout.simple_list_item_1);
ListView lista = (ListView)findViewById(R.id.listElem);
lista.setAdapter(adaptador);
```

117

Controles de selección (12)

- **Lista: *ListView***

Pero podemos crear listas más sofisticadas, en las que cada elemento contenga más de un dato e incluso imágenes, *checkbox*, etc.

Para ello tendríamos que tener los datos de cada elemento almacenados en alguna estructura (*array*, clase, BD) y habría que crear un *layout* con la estructura de cada elemento.

118

Controles de selección (13)

- **Lista: *ListView***

Ejemplo: lista con título y subtítulo

En primer lugar se define una clase "Titular" con la estructura de los elementos.

```
public class Titular {
    private String titulo;
    private String subtitulo;
    public Titular(String tit, String sub){
        titulo = tit;
        subtitulo = sub; }
    public String getTitulo(){
        return titulo; }
    public String getSubtitulo(){
        return subtitulo; }
}
```

119

Controles de selección (14)

- **Lista: *ListView***

A continuación creamos un *layout* con la estructura de cada elemento de la lista: listitem_titular.xml.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
    <TextView android:id="@+id/itemTitulo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="20sp" />
    <TextView android:id="@+id/itemSubTitulo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="normal"
        android:textSize="12sp" />
</LinearLayout>
```

120

Controles de selección (15)

- **Lista: *ListView***

Creamos un adaptador "AdaptadorTitular" a partir de *ArrayAdapter*, para establecer como mostrar los datos de cada elemento.

```
public class AdaptadorTitular extends ArrayAdapter<Titular> {
  private ArrayList<Titular> datos;
  public AdaptadorTitular(Context c, ArrayList<Titular> items){
     super(c, R.layout.listitem_titular, items);
     datos = items;
  }
```

El constructor simplemente llama al constructor del padre (*ArrayAdapter*) y almacena el *ArrayList* de datos recibido.

121

Controles de selección (16)

- **Lista: *ListView***

El método *getView* será llamado cada vez que hay que mostrar un elemento de la lista, y establecerá el dato a mostrar en cada control del *layout*.

```
public View getView(int pos, View vista, ViewGroup parent) {
  LayoutInflater inflater = LayoutInflater.from(getContext());
  View item=inflater.inflate(R.layout.listitem_titular, null);
  TextView titu=(TextView)item.findViewById(R.id.itemTitulo);
  titu.setText(datos.get(pos).getTitulo());
  TextView sub=(TextView)item.findViewById(R.id.itemSubTitulo);
  sub.setText(datos.get(pos).getSubtitulo());
  return item;
}
```

122

Controles de selección (17)

- **Lista: *ListView***

Una vez que tenemos nuestro adaptador definido la forma de crear la lista dentro de la Actividad es igual que en la anterior.

En este caso se ha optado por un *ArrayList*, mucho más versátil que el *Array*.

```
ArrayList items = new ArrayList<Titular>();
items.add(new Titular("Título 1", "Subtítulo largo 1"));
items.add(new Titular("Título 2", "Subtítulo largo 2"));
items.add(new Titular("Título 3", "Subtítulo largo 3"));
/* ... */
AdaptadorTitular adap = new AdaptadorTitular(this, items);
ListView lista = (ListView)findViewById(R.id.listElem);
lista.setAdapter(adap);
```

123

Controles de selección (18)

- **Lista: *ListView***

Por último nos quedaría tratar los eventos sobre la lista, normalmente la selección de un elemento.

```
lista.setOnItemClickListener(new AdapterView.OnItemClickListener(){
  public void onItemClick(AdapterView<?> a, View v, int pos, long id) {
     // Alternativa 1:
     String seleccion = ((Titular)a.getItemAtPosition(pos)).getTitulo();

     // Alternativa 2:
     // String seleccion =
     // ((TextView)v.findViewById(R.id.itemTitulo)).getText().toString();

     /* ... */
    }
});
```

124

Controles de selección (19)

- **Lista: *ListView***

- Actualizar el contenido de una lista

Podemos modificar una lista en curso: añadir elementos, sustituirlos, eliminarlos, etc.

Para ello haremos el cambio que corresponda en el *Adapter* y a continuación llamaremos al método *notifyDataSetChanged*

```
// Insertar un elemento a continuación del primero
Titular nuevo = new Titular("Título 10", "Subtítulo largo 10");
adap.insert(nuevo, 1);
adap.notifyDataSetChanged();
```

125

Controles de selección (20)

- **Lista: *ListView***

- Optimizar una lista

Tal como hemos definido el método *getView*, cada vez que cambien los elementos visibles en pantalla se han de crear todos los objetos de sus *layouts* y llamar a sus métodos *findViewById* con independencia de que ya se hubiera mostrado con anterioridad (por ejemplo al hacer *scroll*), consumiendo una cantidad importante de CPU, memoria y batería.

126

Controles de selección (21)

- **Lista: *ListView***

- Optimizar una lista

Para evitarlo añadiremos a nuestro adaptador una clase interna *ViewHolder* que nos permitirá almacenar referencias a todos los elementos visuales del elemento.

```
public class AdaptadorTitular extends ArrayAdapter<Titular> {
    private ArrayList<Titular> datos;
    static class ViewHolder {
        TextView titulo;
        TextView subtitulo;
    }
    /* ... */
}
```

127

Controles de selección (22)

- **Lista: *ListView***

- Optimizar una lista

La primera vez que se muestre un elemento se crearán todos los objetos necesarios y se guardarán sus referencias en el *ViewHolder*.

Las siguientes veces que sea necesario mostrar el elemento se obtendrá el *ViewHolder* almacenado.

El contenido de los componentes se establecerá siempre a través del *ViewHolder*.

128

Controles de selección (23)

- **Lista: *ListView***

- Optimizar una lista

```
public View getView(int pos, View vista, ViewGroup parent) {
   View item = vista;
   ViewHolder holder;
   if(item == null) {
      LayoutInflater inflater = LayoutInflater.from(getContext());
      item = inflater.inflate(R.layout.listitem_titular, null);
      holder = new ViewHolder();
      holder.titulo = (TextView)item.findViewById(R.id.itemTitulo);
      holder.subtitulo =(TextView)item.findViewById(R.id.itemSubTitulo);
      item.setTag(holder);
   } else {
      holder = (ViewHolder)item.getTag();
   }
   holder.titulo.setText(datos.get(pos).getTitulo());
   holder.subtitulo.setText(datos.get(pos).getSubtitulo());
   return item;
}
```

129

Controles de selección (24)

- ***RecyclerView***

RecyclerView es un componente más flexible y avanzado para la creación de listas que apareció en la versión 5 de Android pero se incluye en la biblioteca de compatibilidad para versiones anteriores.

Este componente tiene mejor rendimiento cuando tenemos que presentar muchos datos ya que requiere del uso de un *ViewHolder* de forma obligatoria.

130

Controles de selección (25)

- ***RecyclerView***

RecyclerView se sustenta en otros componentes, siendo los principales:

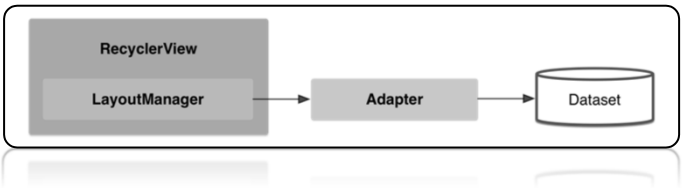
- *RecyclerView.Adapter*: para acceder a los datos utilizaremos un adaptador que heredará de *RecyclerView.Adapter*.
- *RecyclerView.ViewHolder*: el uso de este patrón *ViewHolder* para almacenar las referencias a los objetos será obligatorio.

131

Controles de selección (26)

- ***RecyclerView***

- *LayoutManager*: define la forma en que se van a distribuir los elementos, existiendo algunas predefinidas:
  - *LinearLayoutManager*: lista horizontal o vertical.
  - *GridLayoutManayer*: tabla tradicional.



132

Controles de selección (27)

- ***RecyclerView***

- *ItemDecoration* (opcional): permite definir algunos aspectos de la presentación como marcadores, separadores, etc.
- *ItemAnimator* (opcional): permite definir una animación para los elementos al realizarse determinadas acciones sobre la lista, como añadir, eliminar o intercambiar elementos.

133

Controles de selección (28)

- ***RecyclerView***

A la hora de utilizar un *RecyclerView* no siempre es necesario definir todos los componentes, de hecho lo más habitual es definir el adaptador y el patrón (*holder*), y utilizar alguno de los *LayoutManager* predefinidos.

Si además queremos personalizar nuestra lista podemos añadir algún separador y efecto de animación.

134

Controles de selección (29)

- ***RecyclerView***

Como ejemplo vamos a crear una lista de titulares como la de *ListView*, de manera que podemos reutilizar la clase "Titular" y el *layout* utilizado para representar cada elemento de la lista "listitem_titular.xml".

Hecho esto el primer paso es añadir un *RecyclerView* a nuestro "*activity_main.xml*".

XML

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

135

Controles de selección (30)

- ***RecyclerView***

A continuación crearemos un adaptador extendiendo la clase *RecyclerView.Adapter*.

```
public class AdaptadorTitular extends RecyclerView.Adapter{

    private ArrayList<Titular> datos;

    public AdaptadorTitular(ArrayList<Titular> datos) {
        this.datos = datos;
    }

    /* ... */
}
```

136

Controles de selección (31)

- ***RecyclerView***

Si no aparece la clase *RecyclerView.Adapter*, tendremos que añadir la dependencia correspondiente en el archivo *build.gradle* (app) y a continuación sincronizar el proyecto.

```
dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
   implementation 'com.android.support:appcompat-v7:26.1.0'
   implementation 'com.android.support:recyclerview-v7:26.1.0'
   /* ... */
}
```

137

Controles de selección (32)

- ***RecyclerView***

Nuestro adaptador tendrá que utilizar un *ViewHolder* que extienda a la clase *RecyclerView.ViewHolder.*

```
public static class TitularViewHolder
   extends RecyclerView.ViewHolder {

   private TextView txtTitu;
   private TextView txtSub;

   public TitularViewHolder(View itemView) {
      super(itemView);
      txtTitu =(TextView)itemView.findViewById(R.id.itemTit);
      txtSub = (TextView)itemView.findViewById(R.id.itemSub); }

   public void bindTitular(Titular t) {
      txtTitu.setText(t.getTitulo());
      txtSub.setText(t.getSubtitulo());}
}
```

138

Controles de selección (33)

- ***RecyclerView***

Para acabar con nuestro adaptador tendremos que sobrescribir 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.

139

Controles de selección (34)

- ***RecyclerView***

```
@Override
public TitularViewHolder onCreateViewHolder
  (@NonNull ViewGroup parent, int viewType) {
   View itemView = LayoutInflater.from(parent.getContext()).
     inflate(R.layout.listitem_titular, parent, false);
   TitularViewHolder holder = new TitularViewHolder(itemView);
   return holder;
}
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int pos)
{
    Titular item = datos.get(pos);
    ((TitularViewHolder)holder).bindTitular(item);
}
@Override
public int getItemCount() {
    return datos.size();
}
```

140

Controles de selección (35)

- ***RecyclerView***

Una vez creado el adaptador ya podemos asignárselo al *RecyclerView*.

```
public class MainActivity extends AppCompatActivity {
  private RecyclerView recView;
  private ArrayList<Titular> datos;

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    datos = new ArrayList<Titular>();
    for(int i=0; i<50; i++) {
      datos.add(new Titular("Título " + i, "Sub item " + i)); }

    recView = (RecyclerView) findViewById(R.id.recView);
    recView.setHasFixedSize(true);
    final AdaptadorTitular adap = new AdaptadorTitular(datos);
    recView.setAdapter(adap);
}
```

141

Controles de selección (36)

- ***RecyclerView***

Nos faltaría establecer el *LayoutManager* que define como se distribuyen los elementos.

- *LinearLayoutManager:*

```
recView.setLayoutManager(new
  LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false));
```

- *GridLayoutManager:*

```
recView.setLayoutManager(new GridLayoutManager(this,3));
```

142

Controles de selección (37)

- ***RecyclerView***

LinearLayoutManager

GridLayoutManager

143

Controles de selección (38)

- ***RecyclerView***

Opcionalmente, podemos añadir líneas de separación y animaciones para las operaciones con los elementos.

```
// Añadir separadores
recView.addItemDecoration(new
DividerItemDecoration(this,DividerItemDecoration.VERTICAL));

// Añadir animacion
recView.setItemAnimator(new DefaultItemAnimator());
```

Ejemplos de animaciones:

https://github.com/wasabeef/recyclerview-animators

144

Controles de selección (39)

- ***RecyclerView***

Para probar las animaciones vamos a añadir un botón añadir que agregará un elemento al principio de la lista.

```
// Inserta un nuevo elemento al principio de la lista
Button btnInsertar = (Button)findViewById(R.id.botAdd);
btnInsertar.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      datos.add(1, new Titular("Nuevo titulo", "Subtitulo"));
      adap.notifyItemInserted(1);
    }
});
```

145

Controles de selección (40)

- ***RecyclerView***

En cuanto a la gestión de los eventos de la propia lista, *RecyclerView* no incluye un evento *OnItemClick* por lo que será la propia vista del elemento la que tendrá que gestionarlo.

Existen varias formas de hacerlo, por ejemplo aprovechar la creación del *ViewHolder* para asignar a su vista el evento *onClick*, para ello nuestro adaptador implementará la interfaz *OnClickListener*.

146

Controles de selección (41)

- ***RecyclerView***

```
public class AdaptadorTitular
  extends RecyclerView.Adapter<AdaptadorTitular.TitularViewHolder>
  implements View.OnClickListener {

   private ArrayList<Titular> datos;
   private View.OnClickListener listener;

   public TitularViewHolder onCreateViewHolder(ViewGroup parent,
     int viewType) {
     View itemView = LayoutInflater.from(parent.getContext()).
       inflate(R.layout.listitem_titular, parent, false);
     itemView.setOnClickListener(this);
     TitularViewHolder holder = new TitularViewHolder(itemView);
     return holder;
   }
   /* */
   public void setOnClickListener (View.OnClickListener listen){
      this.listener = listen; }
   public void onClick (View v) {
       if(listener != null)
          listener.onClick(v); }
```

147

Controles de selección (42)

- ***RecyclerView***

Tras modificar el adaptador, ya podemos gestionar los eventos desde la actividad principal.

```
final AdaptadorTitular adap = new AdaptadorTitular(datos);
adap.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
       datos.remove(recView.getChildAdapterPosition(v));
       adap.notifyDataSetChanged();
    }
});
recView.setAdapter(adap);
```

148

Controles de selección (43)

- ***CardView***

CardView aparece también en Android 5 y nos permite presentar la información en forma de tarjetas en las que podremos personalizar fondo, contorno, elevación, etc.

Suele utilizarse en combinación con una lista *ListView* o *RecyclerView*.

149

Controles de selección (44)

- ***CardView***

Podemos crear un *CardView* agregándolo a nuestro *Layout*, y añadiendo en su interior todos los componentes que deseemos.

Para crear una lista de elementos *CardView*, basta con utilizar este componente en la definición del *Layout* de los ítem de la lista.

XML

```
<androidx.cardview.widget.CardView
  android:id="@+id/card"
  android:layout_width="match_parent"
  android:layout_height="200dp"
  app:cardBackgroundColor="@android:color/holo_orange_light" />
```

150

Controles de selección (45)

- ***CardView***

CardView es una extensión de *FrameLayout*, de manera que podemos apilar elementos.

Por ejemplo podemos utilizar un *ImageView* para establecer una imagen de fondo, y sobre ella colocar texto u otros controles.

Además podemos personalizar su apariencia a través de las propiedades:

- *cardCornerRadius*: redondear esquinas.
- cardElevation: elevación de la tarjeta.

151

Controles de selección (46)

- ***CardView***

Podemos encontrarnos con que el aspecto de la tarjeta difiere de una versión de Android a otra en cuanto a la sombra y los bordes. Para reducir esas diferencias podemos utilizar las propiedades:

- *cardUseCompatPadding*: añade un *padding* extra para que se vea la sombra.
- *cardPreventCornerOverlap*: evita el margen adicional en las versiones anteriores.

152

Controles de selección (47)

- ***CardView*: ejemplo**

XML

```
<androidx.cardview.widget.CardView
    android:id="@+id/card"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    app:cardPreventCornerOverlap="true"
    app:cardUseCompatPadding="true"
    app:cardElevation="8dp"
    app:cardCornerRadius="20dp">

    <ImageView
        android:id="@+id/imgCard"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:adjustViewBounds="false"
        android:scaleType="fitXY"
        android:src="@drawable/leon" />
```

153

Controles de selección (48)

- ***CardView*: ejemplo**

XML

```
    <TextView
        android:id="@+id/textCard"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:paddingVertical="6dp"
        android:paddingHorizontal="10dp"
        android:text="León"
        android:textColor="#ffffff"
        android:background="#8c000000"
        android:textStyle="bold"
        android:textSize="30sp" />

</androidx.cardview.widget.CardView>
```

154