

tema3_componentes_avanzados

Tema 3. Componentes avanzados

ÍNDICE.

1. COMPONENTES AVANZADOS.

1.1. Spinner

1.2. ListView

1.3. GridView

1.4. AutoCompleteTextView

1.5. CardView

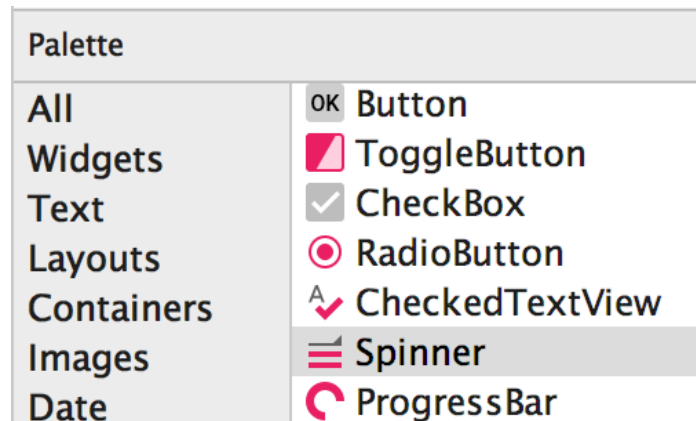
1.6. RecyclerView

1.7. Componentes definidos por el usuario

1.1 Spinner

Spinner

Es un control que muestra los datos como una lista desplegable de opciones donde poder elegir.



En el layout añadiremos el código indicado o arrastraremos el control:

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

A continuación, en el fichero *.java* o *.kt* indicaremos lo siguiente:

1. Definiremos el vector de datos que se desplegará sobre el Spinner.
2. Crearemos el adaptador. En este caso, vamos a utilizar un adaptador del tipo `ArrayAdapter` que viene predefinido por Android. Estableceremos la relación entre el vector de datos y el adaptador.
3. Situaremos sobre el control Spinner el adaptador creado.

En Java:

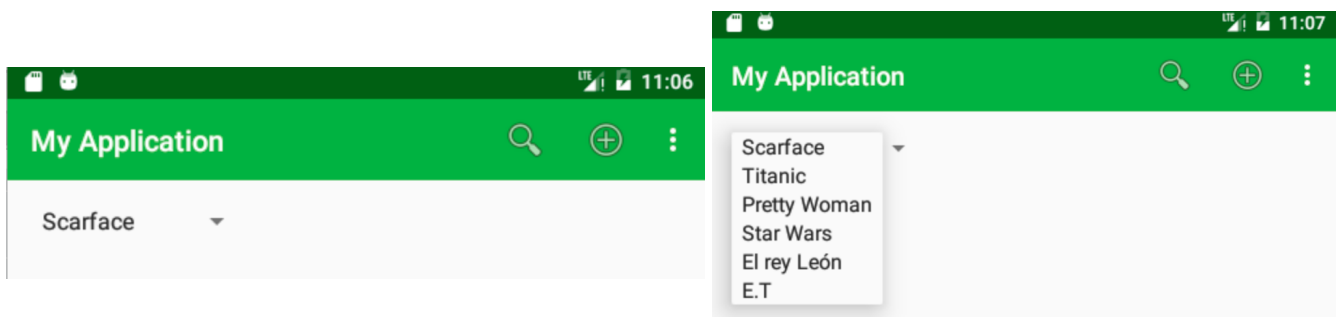
```
String[] peliculas = new String[]{"Scarface", "Titanic", "Pretty  
ArrayAdapter<String> adaptador = new ArrayAdapter<String> (this,  
  
Spinner list_peliculas= (Spinner) findViewById(R.id.spinner);  
list_peliculas.setAdapter(adaptador);
```

```
adaptador.setDropDownViewResource(android.R.layout.simple_spinner
```

En Kotlin:

```
val peliculas = arrayOf("Scarface", "Titanic", "Pretty Woman", "S
val adaptador = ArrayAdapter(this, android.R.layout.simple_spinne
val list_peliculas = findViewById(R.id.spinner) as Spinner
list_peliculas.adapter = adaptador
adaptador.setDropDownViewResource(android.R.layout.simple_spinner
```

El resultado será:



Para mostrar en pantalla el ítem del Spinner seleccionado, añadiremos el siguiente código:

En Java:

```
Spinner list_peliculas= (Spinner) findViewById(R.id.spinner);
list_peliculas.setOnItemClickListener(new AdapterView.OnItemClickListener
@Override
    public void onItemClick(AdapterView<?> adapterView, View vie
        Toast.makeText(MainActivity.this, parent.getItemAtPosition(po
    }
});
```

En Kotlin:

```
val list_peliculas = findViewById(R.id.spinner) as Spinner
```

```
list_peliculas.setOnItemSelectedListener(object : AdapterView.OnI
    override fun onItemSelected(parent: AdapterView<*>?, view: Vi
        Toast.makeText(applicationContext, parent!!.getItemAtPosi
    }
    override fun onNothingSelected(parent: AdapterView<*>?) {
        TODO("not implemented") //To change body of created funct
    }
})
```



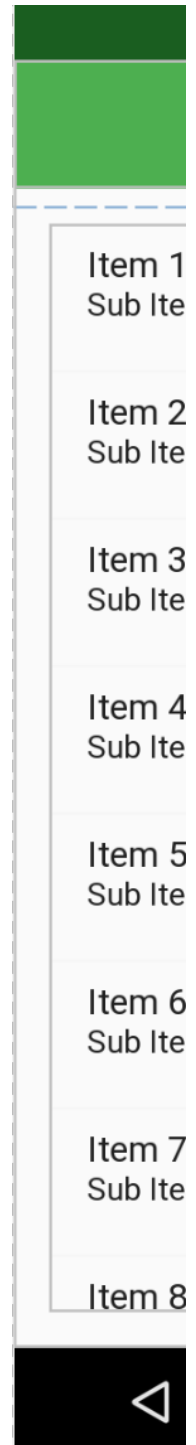
1.2 ListView

ListView

Este componente presenta los datos como una lista fija con orientación vertical.

En el layout principal *activity_main.xml*, añadiríamos el siguiente código:

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



A continuación, igual que hemos hecho en el Spinner, en el fichero *.java* o *.kt* indicaremos lo siguiente:

1. Definiremos el vector de datos que se desplegará sobre el ListView..
2. Crearemos el adaptador. En este caso, vamos a utilizar un adaptador del tipo ArrayAdapter que viene predefinido por Android. Estableceremos la relación entre el vector de datos y el adaptador.
3. Situaremos sobre el control ListView el adaptador creado.

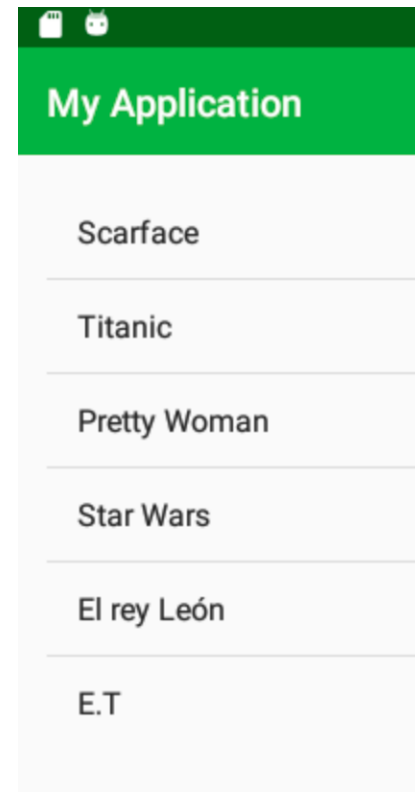
En Java:

```
String[] peliculas = new String[]{"Scarface", "Titanic", "Pretty  
ArrayAdapter<String> adaptador = new ArrayAdapter<String> (this,  
ListView list_peliculas= (ListView) findViewById(R.id.listview);  
list_peliculas.setAdapter(adaptador);
```

En Kotlin:

```
val peliculas = arrayOf("Scarface", "Titanic", "Pretty Woman", "S  
val adaptador = ArrayAdapter(this, android.R.layout.simple_list_i  
val list_peliculas = findViewById(R.id.listview) as ListView  
list_peliculas.adapter = adaptador
```

El resultado será:



Para mostrar en pantalla el ítem del ListView seleccionado, añadiremos el siguiente código:

En Java:

```
ListView list_peliculas= findViewById(R.id.listview);

list_peliculas.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v, int pos) {
        Toast.makeText(MainActivity.this, parent.getItemAtPosition(pos), Toast.LENGTH_SHORT).show();
    }
});
```

En Kotlin:

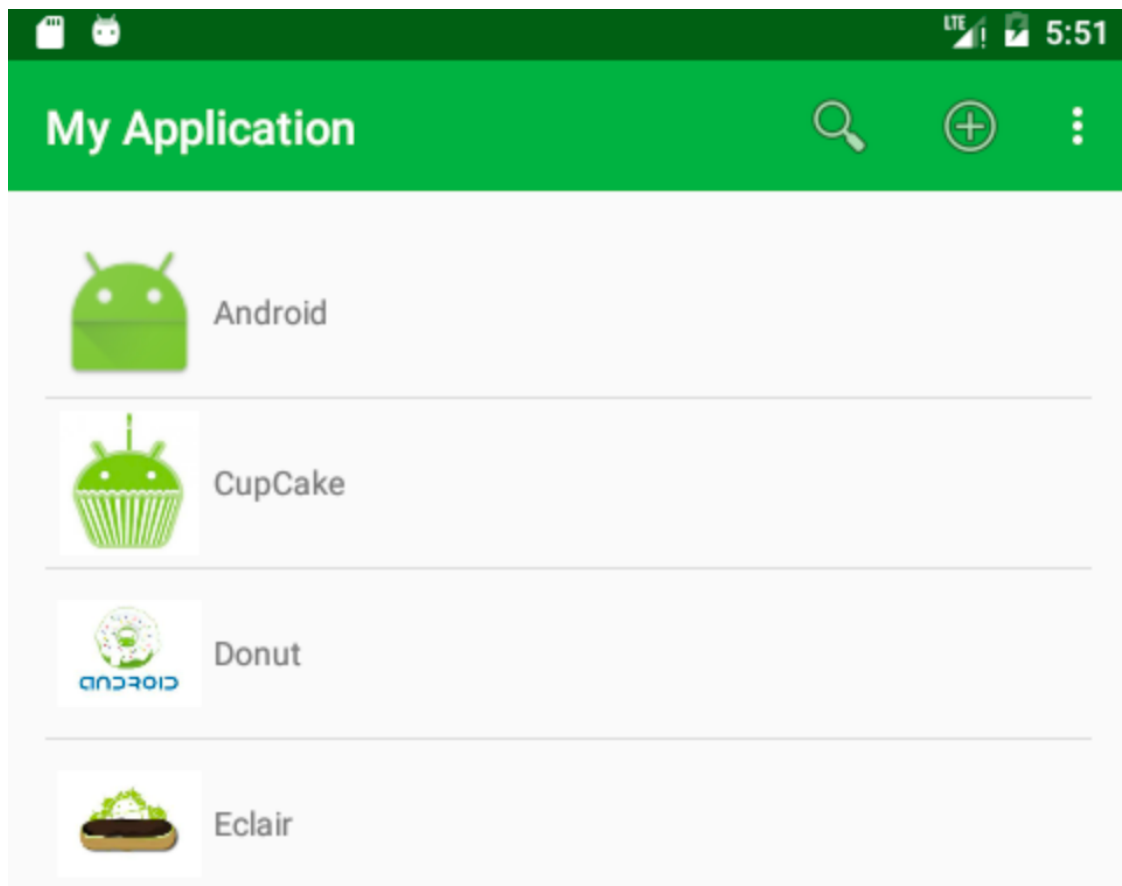
```
val list_peliculas= findViewById(R.id.listview) as ListView

list_peliculas.setOnItemClickListener(object : AdapterView.OnItemClickListener {
    override fun onItemClick(parent: AdapterView<*>, v: View, pos: Int) {
        Toast.makeText(applicationContext, parent.getItemAtPosition(pos), Toast.LENGTH_SHORT).show()
    }
})
```

Pero, ¿y si quisiéramos presentar en cada ítem del ListView, una imagen y un texto, en lugar de únicamente una cadena de texto (string)? En ese caso, como Android no proporciona ningún adaptador que se adecue a nuestras necesidades, tendríamos que definir nuestro propio adaptador.

Ejemplo. ListView con adaptador definido por el usuario.

Queremos diseñar un ListView que muestre en cada ítem: una imagen (ImageView) con el logotipo de la versión de Android, y un texto (TextView) donde se especifique el nombre de la versión. El resultado a obtener será:



Para ello, realizamos los siguientes pasos:

- En el layout principal *activity_main.xml* insertaremos el ListView, tal y como hemos hecho anteriormente. Este layout será cargado en el fichero *MainActivity.java* o *MainActivity.kt*
- Crearemos un nuevo layout (*list_item.xml*) que contendrá los elementos de un único ítem del ListView. En este caso, un ImageView y un TextView.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res
```



```

    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/idLogo"
        android:layout_width="72dp"
        android:layout_height="72dp"
        android:padding="5dp"
        android:src="@mipmap/donut"/>

    <TextView
        android:id="@+id/idNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_toRightOf="@id/idLogo"
        android:text="Donut" />

</RelativeLayout>

```

- A continuación, crearemos una clase *AndroidVersion.java* para manipular los elementos del ítem del ListView. En este caso, un texto (String) y una imagen (int). Los elementos del vector de datos que cargaremos posteriormente en el *MainActivity.java* serán del tipo *AndroidVersion*.

```

public class AndroidVersion
{
    String nombre;
    int logo;

    public AndroidVersion(String nombre, int logo) {
        super();
        this.nombre = nombre;
        this.logo = logo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {

```

```

        this.nombre = nombre;
    }
    public int getLogo() {
        return logo;
    }
    public void setLogo(int logo) {
        this.logo = logo;
    }
}

```

En Kotlin:

```
class AndroidVersion(var nombre: String, var logo: Int)
```

- Ahora es momento de diseñar el adaptador *VersionesAdapter.java*, donde realizaremos las siguientes funciones:
 - Extenderemos de *BaseAdapter*, el adaptador más sencillo de los predefinidos por Android (aunque también podríamos extender de *ArrayAdapter*).
 - Crearemos el constructor, que tendrá dos parámetros: el contexto y el vector de datos.
 - Definimos los métodos, entre los que cabe destacar el método **getView()**. Este método será el encargado de inflar el ítem del ListView con su contenido. El ítem que se dibujará será el elemento del vector correspondiente a la posición **position**.

```

public class VersionesAdapter extends BaseAdapter {
    private LayoutInflater mInflater;
    private ArrayList<AndroidVersion> versiones;

    public VersionesAdapter(Context context, ArrayList<AndroidVers
        super();
        this.mInflater = LayoutInflater.from(context);
        this.versiones = vers;
    }

    public int getCount() {
        return versiones.size();
    }
}

```

```

    }
    public AndroidVersion getItem(int position) {
        return versiones.get(position);
    }
    public long getItemId(int position) {
        return position;
    }

    public View getView(int position, View convertView, ViewGroup
        convertView = mInflater.inflate(R.layout.list_item, null)
        TextView hNombre = (TextView) convertView.findViewById(R.
        ImageView hImage = (ImageView) convertView.findViewById(R

        AndroidVersion version = getItem(position);
        hNombre.setText(version.getNombre());
        hImage.setImageResource(version.getLogo());
        return convertView;
    }
}

```

- Finalmente, en el fichero *MainActivity.java* cargaremos el vector de datos, donde los elementos de dicho vector serán del tipo *AndroidVersion*:

```

// Nuestra colección de datos
ArrayList<AndroidVersion> versiones = new ArrayList<AndroidVersio
versiones.add(new AndroidVersion("Android", R.mipmap.launcher));
versiones.add(new AndroidVersion("CupCake", R.mipmap.cupcake));
versiones.add(new AndroidVersion("Donut", R.mipmap.donut));
versiones.add(new AndroidVersion("Eclair",R.mipmap.eclair));

ListView lista_versiones= (ListView) findViewById(R.id.listview);

// Creamos el adaptador con el vector de datos
VersionesAdapter adapter = new VersionesAdapter(MainActivity.this

// Situamos sobre el ListView el adaptador.
lista_versiones.setAdapter(adapter);

//Al hacer click sobre un ítem del ListView obtendremos un Toast

```

```

lista_versiones.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
        Toast.makeText(MainActivity.this, "" + position, Toast.LENGTH_SHORT).show();
    }
});

```

En Kotlin:

```

val versiones = ArrayList<AndroidVersion>()
versiones.add(AndroidVersion("Android", R.mipmap.logo))
versiones.add(AndroidVersion("CupCake", R.mipmap.cupcake))
versiones.add(AndroidVersion("Donut", R.mipmap.donut))
versiones.add(AndroidVersion("Eclair", R.mipmap.eclair))

val lista_versiones = findViewById(R.id.listview) as ListView
val adaptador = VersionesAdapter(this@MainActivity, versiones)
lista_versiones.adapter=adaptador

lista_versiones.setOnItemClickListener(object : AdapterView.OnItemClickListener() {
    override fun onItemClick(parent: AdapterView<*>, v: View, position: Int, id: Long) {
        Toast.makeText(this@MainActivity, "" + position, Toast.LENGTH_SHORT).show();
    }
})

```

Con todo esto, obtendríamos el resultado esperado. Sin embargo, cabe replantearse la forma en que se dibujan los ítems del ListView, ya que si el número de ítems es pequeño, no hay problemas de eficiencia, pero si el número de ítems es elevado y éste presenta además una estructura compleja, el consumo de batería se dispara. Esto es debido a que cuando se hace scroll sobre el ListView, el sistema debe dibujar los ítems que se visualizan en pantalla y destruir todos aquellos que dejan de estar visibles, y así continuamente. El consumo de batería se eleva en exceso realizando la función de dibujar y destruir ítems (layouts).

¿Cómo podríamos por tanto, mejorar la eficiencia de nuestro dispositivo?

Podríamos reescribir el método **getView()**, de manera que, si un ítem o layout ya ha sido dibujado, en lugar de destruirlo podamos almacenarlo y recuperarlo cuando sea necesario dibujar un nuevo ítem. Con esto conseguiríamos aprovechar los layouts ya

dibujados que han quedado "obsoletos" o en desuso para dibujar nuevos layouts.

Para ello, Android proporciona:

- la variable **convertView**, la cual almacenará cualquier layout que haya quedado en desuso. Así pues, si la variable no es nula podemos reutilizar un layout obsoleto y ahorrarnos el trabajo de inflarlo.
- y el contenedor **ViewHolder**, clase que desarrollaremos para almacenar controles ya buscados (findViewById), y así evitar tener que buscarlos nuevamente. Android permite identificar los controles mediante el uso de una etiqueta (tag). Por tanto, podremos usar los métodos setTag() y getTag().

Con estas mejoras el código optimizado quedará:

```
public View getView(int position, View convertView, ViewGroup par
    ViewHolder holder = null;
    if (convertView == null) {
        convertView = mInflater.inflate(R.layout.list_item, null)
        holder = new ViewHolder();
        holder.hNombre = (TextView) convertView.findViewById(R.id
        holder.hImage = (ImageView) convertView.findViewById(R.id
        convertView.setTag(holder);
    }
    else {
        holder = (ViewHolder) convertView.getTag();
    }
    AndroidVersion version = getItem(position);
    holder.hNombre.setText(version.getNombre());
    holder.hImage.setImageResource(version.getLogo());
    return convertView;
}

class ViewHolder {
    TextView hNombre;
    ImageView hImage;
}
```

En Kotlin, el adaptador VersionesAdapter extendiendo de ArrayAdapter quedaría::

```
class VersionesAdapter(context: Context, private val versiones: A
{
    private val mInflater: LayoutInflater

    init {
        this.mInflater = LayoutInflater.from(context)
    }

    override fun getCount(): Int {
        return versiones.size
    }

    override fun getItem(position: Int): AndroidVersion? {
        return versiones[position]
    }

    override fun getItemId(position: Int): Long {
        return position.toLong()
    }

    override fun getView(position: Int, convertView: View?, parent
        var convertView = convertView
        var holder: ViewHolder? = null

        if (convertView == null) {
            convertView = mInflater.inflate(R.layout.list_item, null)
            holder = ViewHolder()
            holder.hNombre = convertView.findViewById(R.id.idNombre) a
            holder.hImage = convertView.findViewById(R.id.idLogo) as I
            convertView.tag = holder
        }
        else {
            holder = convertView!!.tag() as ViewHolder?
        }

        val deporte = getItem(position)
        holder!!.hChb!!.text = deporte!!.nombre
        holder!!.hChb!!.isChecked = deporte!!.pulsado
        holder!!.hImage!!.setImageResource(deporte!!.logo)
        return convertView!!
```

```
}

internal inner class ViewHolder {
    var hNombre: TextView? = null
    var hImage: ImageView? = null
}
}
```

Nota. Adaptador VersionesAdapter.java extendiendo de ArrayAdapter.

```
public class VersionesAdapter extends ArrayAdapter<AndroidVer
    private LayoutInflater mInflater;
    private ArrayList<AndroidVersion> versiones;


    public VersionesAdapter(Context context, ArrayList<Androi
        super(context, R.layout.list_item, vers);
        this.mInflater = LayoutInflater.from(context);
        this.versiones = vers;
    }

    public int getCount() {
        return versiones.size();
    }
    public AndroidVersion getItem(int position) {
        return versiones.get(position);
    }
    public long getItemId(int position) {
        return position;
    }

    public View getView(int position, View convertView, ViewG
        ViewHolder holder = null;
        if (convertView == null) {
            convertView = mInflater.inflate(R.layout.list_ite
            holder = new ViewHolder();
            holder.hNombre = (TextView) convertView.findViewB
```

```
        holder.hImage = (ImageView) convertView.findViewById(R.id.hImage);
        convertView.setTag(holder);
    }
    else {
        holder = (ViewHolder) convertView.getTag();
    }
    AndroidVersion version = getItem(position);
    holder.hNombre.setText(version.getNombre());
    holder.hImage.setImageResource(version.getLogo());
    return convertView;
}

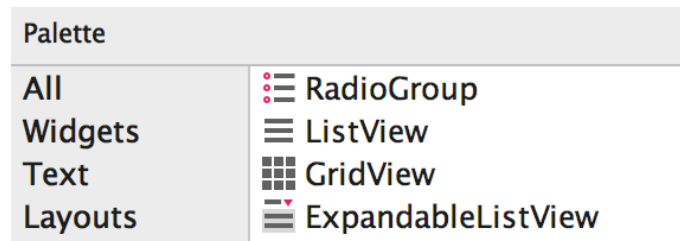
class ViewHolder {
    TextView hNombre;
    ImageView hImage;
}
}
```



1.3 GridView

GridView

Este componente presenta los datos de forma tabular.



En el layout principal *activity_main.xml*, añadiríamos el siguiente código o arrastraríamos el control:

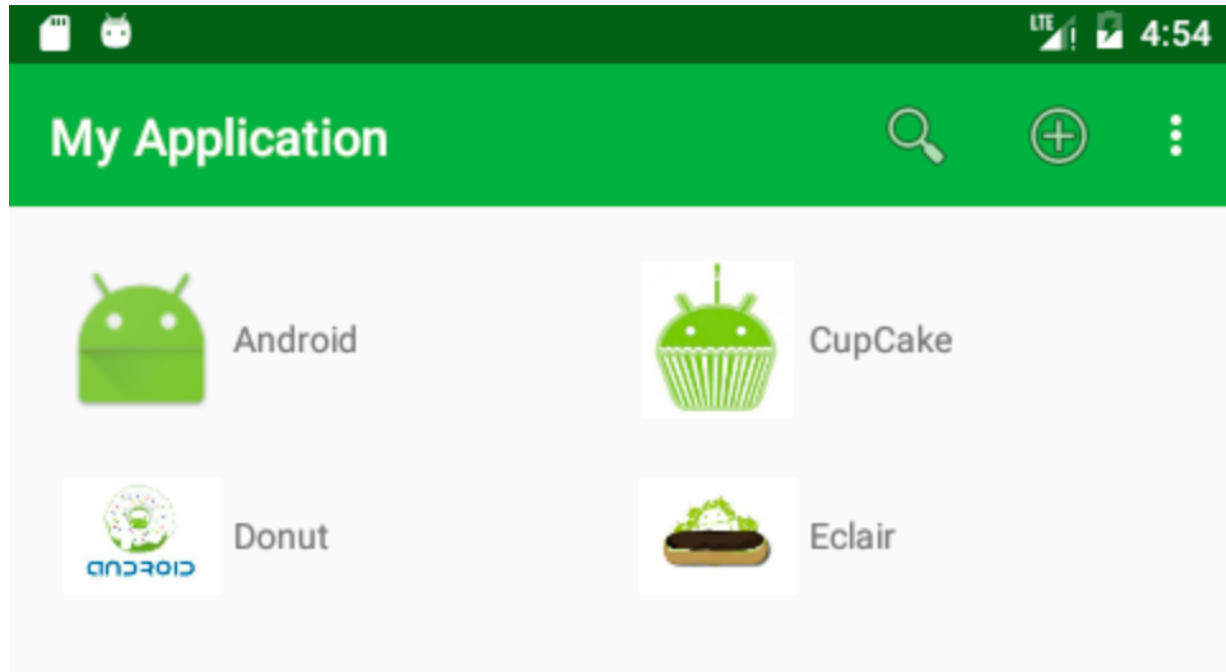
```
<GridView
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:numColumns="auto_fit"
    android:verticalSpacing="5dp"
    android:horizontalSpacing="5dp"
    android:stretchMode="columnWidth"/>
```

Entre las propiedades a destacar tenemos:

- numColumns: podemos indicar el número exacto de columnas que tendrá la tabla, o bien *auto_fit* donde el número de columnas de la tabla se adaptará al número de elementos a representar y al espacio disponible.
- verticalSpacing: espacio entre filas.
- horizontalSpacing: espacio entre columnas.
- stretchMode: espacio disponible o sobrante a la derecha de la pantalla, será ocupado a partes iguales por las columnas si ponemos la propiedad

a *columnWidth*, o bien si ponemos *spacingWidth* esté espacio será ocupado a partes iguales por los espacios entre columnas.

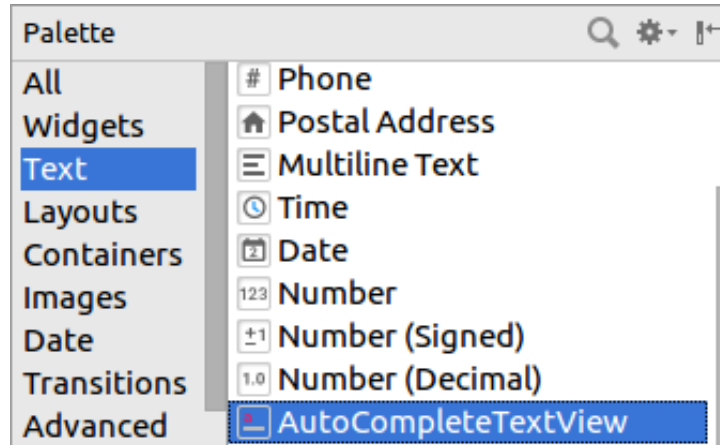
Con el código descrito anteriormente y utilizando el mismo adaptador que hemos desarrollado para el ListView, obtendremos el siguiente resultado:



1.4 AutoCompleteTextView

AutoCompleteTextView

Es un control que muestra un cuadro de texto con sugerencias.



En el layout añadiremos el código indicado o arrastraremos el control:

```
<AutoCompleteTextView
    android:id="@+id/autoCompleteTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

A continuación, en el fichero `.java` indicaremos lo siguiente:

1. Definiremos el vector de datos que aparecerá como sugerencias sobre el cuadro de texto.
2. Crearemos el adaptador. En este caso, vamos a utilizar un adaptador del tipo `ArrayAdapter` que viene predefinido por Android. Estableceremos la relación entre el vector de datos y el adaptador.
3. Situaremos sobre el control `AutoCompleteTextView` el adaptador creado.
4. El método `setThreshold(int)` permite establecer con cuantos caracteres escritos (int) aparece el cuadro de sugerencias. En el ejemplo, con 1 carácter.

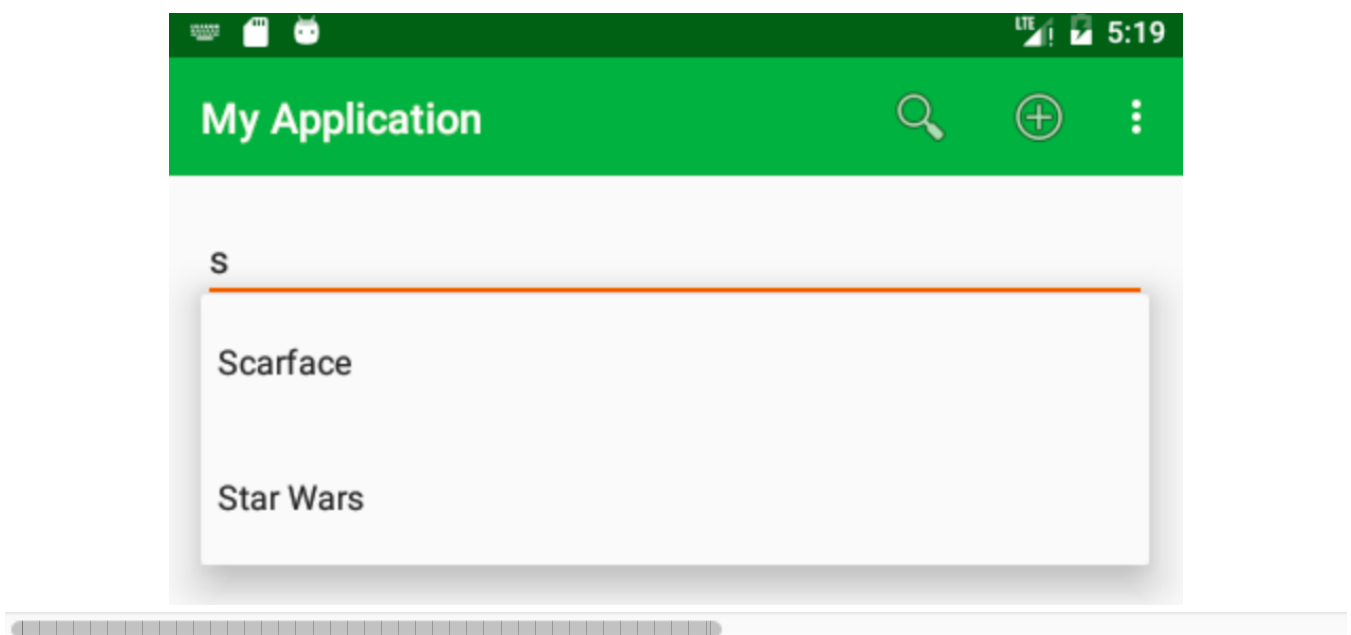
```
String[] peliculas = new String[]{"Scarface", "Titanic", "Pretty  
AutoCompleteTextView list_peliculas= (AutoCompleteTextView) findV
```

```
ArrayAdapter<String> adaptador = new ArrayAdapter<String> (this,  
list_peliculas.setAdapter(adaptador);  
list_peliculas.setThreshold(1);
```

En Kotlin:

```
val peliculas = arrayOf("Scarface", "Titanic", "Pretty Woman", "S  
val list_peliculas = findViewById(R.id.autoCompleteTextView) as A  
val adaptador = ArrayAdapter<String> (this, android.R.layout.simp  
list_peliculas.adapter = adaptador  
list_peliculas.threshold = 1
```

El resultado será:

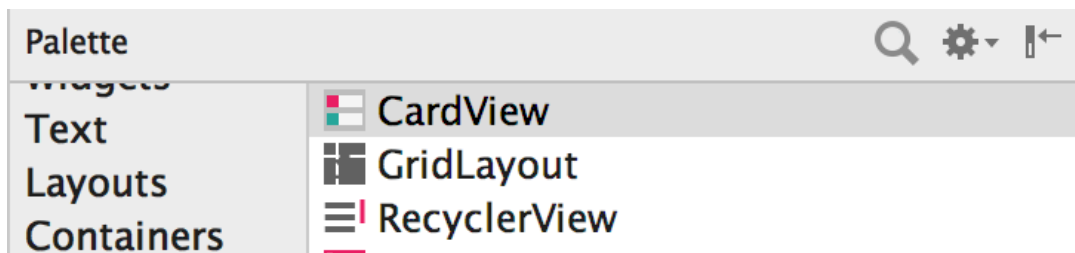


1.5 CardView

CardView

Este componente presenta el aspecto de una tarjeta, con elevación y esquinas redondeadas. Realiza función de contenedor o layout. Su comportamiento es exactamente igual que el del `FrameLayout`.

Necesita para su funcionamiento tener enlazada la librería `'androidx.cardview:cardview:1.0.0'` en el fichero `build.gradle` de la aplicación.



En el layout insertaremos el siguiente código o arrastraremos el control. Añadiremos las propiedades `cardCornerRadius`, que define el radio del borde redondeado, la elevación de la tarjeta, establecido en la propiedad `cardElevation` y pondremos la propiedad `cardUseCompatPadding` a `true`, para evitar que se visualice la sombra con ángulo recto por debajo de la tarjeta.

```
<androidx.cardview.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="200dp"
    app:cardCornerRadius="12dp"
    app:cardElevation="10dp"
    app:cardUseCompatPadding="true">
```

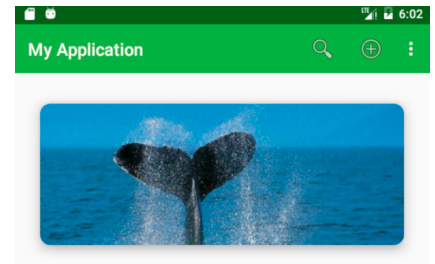
Vamos a crear como ejemplo, una tarjeta (`CardView`) que contenga una imagen (`ImageView`) en su interior. En el layout añadiremos el código indicado, obteniendo el siguiente resultado:

```
<androidx.cardview.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="200dp"
```

```
app:cardCornerRadius="12dp"  
app:cardElevation="10dp"  
app:cardUseCompatPadding="true">
```

```
<ImageView  
    android:id="@+id/imageView"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:scaleType="centerCrop"  
    app:srcCompat="@mipmap/whale" />
```

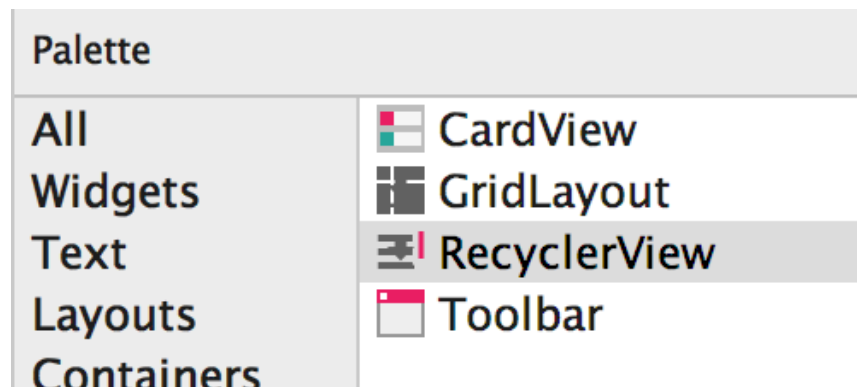
```
</android.support.v7.widget.CardView>
```



1.6 RecyclerView

RecyclerView

Este componente viene a substituir al ListView y al GridView, ya que con un único control podemos emular los dos anteriores.



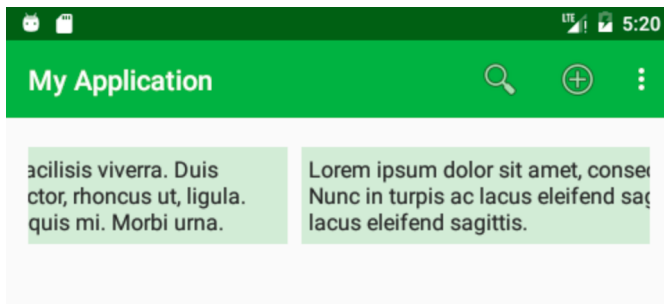
Necesita enlazar en el fichero **build.gradle** la siguiente librería para poder funcionar: **implementation 'androidx.recyclerview:recyclerview:1.0.0'**

El RecyclerView es un control que por si mismo aporta poca funcionalidad. La forma en que se presentan los datos se delega sobre el LayoutManager. Tenemos disponible tres tipos de LayoutManager:

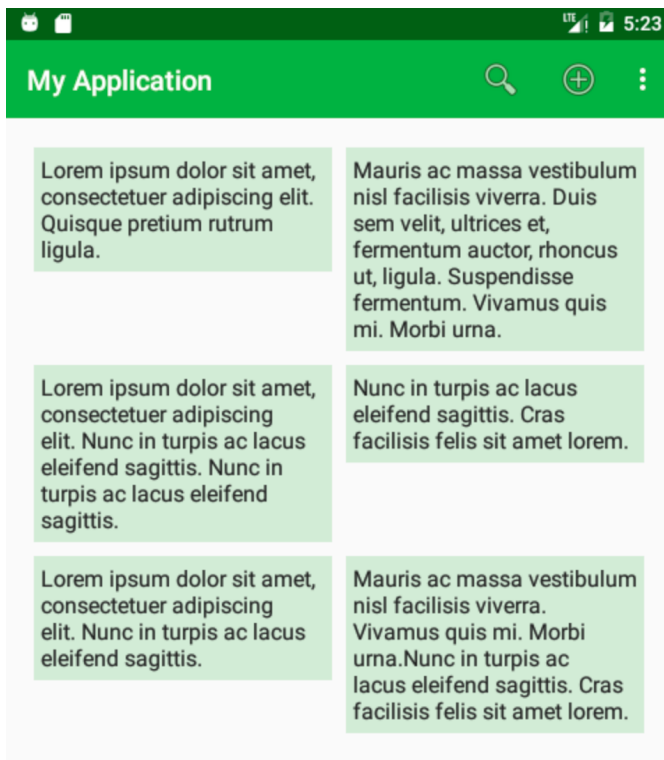
- **LinearLayoutManager**: presenta los datos en forma de lista horizontal, como el antiguo control Gallery, o vertical, como el ListView..
- **GridLayoutManager**: presenta los datos en forma de tabla con todas las celdas de igual tamaño. Similar al componente GridView.
- **StaggeredGridLayoutManager**: como el caso anterior, pero las celdas pueden tener tamaños diferentes.

LinearLayoutManager horizontal

LinearLayoutManager vertical



GridLayoutManager



StaggeredGridLayoutManager



En el layout principal *activity_main.xml*, añadiríamos el siguiente código o arrastraríamos el control:

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

```
android:layout_height="match_parent" />
```

El control RecyclerView necesita utilizar un adaptador para cargar los datos, y éste difiere bastante del que hemos utilizado hasta ahora. Por tanto, vamos a ver un ejemplo:

Ejemplo. RecyclerView con LinearLayoutManager vertical

El resultado a obtener será el siguiente:



Para ello, realizamos los siguientes pasos:

- En el layout principal *activity_main.xml* insertaremos el RecyclerView, tal y como hemos hecho anteriormente. Este layout será cargado en el fichero *MainActivity.java*.

- Crearemos un nuevo layout (*item_cards.xml*) que contendrá los elementos de un único ítem del RecyclerView. En este caso, un TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <TextView
        android:id="@+id/txt1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="5dp"
        android:background="@color/colorPrimaryLight"
        android:textColor="#212121"
        android:textSize="16sp" />

</LinearLayout>
```

- A continuación, crearemos una clase *Tarjeta.java* para manipular los elementos del ítem del RecyclerView. En este caso, únicamente un texto. Los elementos del vector de datos que cargaremos posteriormente en el *MainActivity.java* serán del tipo Tarjeta.

```
public class Tarjeta {
    private int texto;

    public Tarjeta(int text){
        texto = text;
    }
    public int getTexto(){
        return texto;
    }
}
```

En Kotlin (*Tarjeta.kt*):

```
class Tarjeta(val texto: Int)
```

- Ahora es momento de diseñar el adaptador *CardsAdapter.java*, donde realizaremos las siguientes funciones:
 - Extenderemos de *RecyclerView.Adapter*.
 - Crearemos una nueva clase dentro del adaptador que extenderá de *RecyclerView.ViewHolder*. Recordad que el ViewHolder es un contenedor o mantenedor de componentes ya dibujados en cada ítem. En este caso, el ítem sólo contiene un TextView.
 - Definiremos los métodos, entre los que caben destacar:
 - el método **onCreateViewHolder**. Este método será el encargado de inflar el ítem del RecyclerView con su contenido de forma optimizada. Es decir, se reutilizarán layouts en desuso y se evitará buscar componentes que ya estén almacenados en el ViewHolder.
 - el método **onBindViewHolder**. Este método será el encargado de actualizar los datos en el ítem del RecyclerView. El ítem que se dibujará será el indicado en la posición *pos*.

```
public class CardsAdapter extends RecyclerView.Adapter<CardsAdapt

private static ArrayList<Tarjeta> items;
public CardsAdapter(ArrayList<Tarjeta> items) {
    this.items = items;
}

public class TarjViewHolder extends RecyclerView.ViewHolder {
    private TextView texto;

    public TarjViewHolder(View itemView) {
        super(itemView);

        texto = (TextView) itemView.findViewById(R.id.txt1);
    }

    public void bindTitular(Tarjeta t) {
        texto.setText(t.getTexto());
    }
}
```

```

    }

    @Override
    public TarjViewHolder onCreateViewHolder(ViewGroup viewGroup,
        View itemView = LayoutInflater.from(viewGroup.getContext())
        TarjViewHolder tvh = new TarjViewHolder(itemView);
        return tvh;
    }

    @Override
    public void onBindViewHolder(TarjViewHolder viewHolder, int p
        Tarjeta item = items.get(pos);
        viewHolder.bindTitular(item);
    }

    @Override
    public int getItemCount() {
        return items.size();
    }
}

```

En Kotlin (*CardsAdapter.kt*):

```

class CardsAdapter(var items: ArrayList<Tarjeta>) : RecyclerView.

    init {
        this.items = items
    }

    class TarjViewHolder(itemView: View) : RecyclerView.ViewHolde

        private var texto: TextView

        init {
            texto = itemView.findViewById(R.id.txt1)
        }

        fun bindTarjeta(t: Tarjeta) {
            texto.setText(t.texto)
        }
    }

```

```

    }

    override fun onCreateViewHolder(viewGroup: ViewGroup, viewType: Int) {
        val itemView = LayoutInflater.from(viewGroup.context).inflate(R.layout.item_tarjeta, viewGroup, false)
        return TarjViewHolder(itemView)
    }

    override fun onBindViewHolder(viewHolder: TarjViewHolder, pos: Int) {
        val item = items.get(pos)
        viewHolder.bindTarjeta(item)
    }

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

```

Notad como las funciones realizadas en los dos métodos propios del RecyclerView (onCreateViewHolder y onBindViewHolder), son equivalentes a la función del método getView en el componente ListView.

- Finalmente, en el fichero *MainActivity.java* cargaremos el vector de datos, donde los elementos de dicho vector serán del tipo Tarjeta (los textos de cada ítem han sido cargados en el fichero de strings (values/strings.xml)):

```

final ArrayList<Tarjeta> items = new ArrayList<Tarjeta>();
items.add(new Tarjeta(R.string.note1));
items.add(new Tarjeta(R.string.note2));
items.add(new Tarjeta(R.string.note3));
items.add(new Tarjeta(R.string.note4));
items.add(new Tarjeta(R.string.note5));
items.add(new Tarjeta(R.string.note6));

final RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);

//el RecyclerView tendrá tamaño fijo
recyclerView.setHasFixedSize(true);

CardsAdapter adaptador = new CardsAdapter(items);

```

```
recView.setAdapter(adaptador);  
recView.setLayoutManager(new LinearLayoutManager(this, LinearLayo
```

En Kotlin (*MainActivity.kt*):

```
val items = ArrayList<Tarjeta>()  
items.add(Tarjeta(R.string.note1))  
items.add(Tarjeta(R.string.note2))  
items.add(Tarjeta(R.string.note3))  
items.add(Tarjeta(R.string.note4))  
items.add(Tarjeta(R.string.note5))  
items.add(Tarjeta(R.string.note6))  
  
val recView = findViewById<RecyclerView>(R.id.recyclerView)  
  
recView.setHasFixedSize(true)  
  
val adaptador = CardsAdapter(items)  
recView.adapter = adaptador  
recView.layoutManager = LinearLayoutManager(this, LinearLayoutManager
```

En el ListView mediante el uso del método `setOnClickListener`, hemos obtenido un Toast con la posición del elemento seleccionado al hacer click sobre un ítem de dicho componente. Sin embargo, el RecyclerView no lleva implementado por defecto el método `setOnClickListener`, así que tendremos que implementarlo nosotros mismos.

Sobre el adaptador, añadiremos las siguientes sentencias:

```
11 public class CardsAdapter extends RecyclerView.Adapter<CardsAdapter.TarjViewHolder> implements View.OnClickListener {
12
13     private static ArrayList<Tarjeta> items;
14     private View.OnClickListener listener;
15
16     public CardsAdapter(ArrayList<Tarjeta> items) { this.items = items; }
17
18     public class TarjViewHolder extends RecyclerView.ViewHolder {
19         private TextView texto;
20
21         public TarjViewHolder(View itemView) {
22             super(itemView);
23
24             texto = (TextView) itemView.findViewById(R.id.txt1);
25         }
26
27         public void bindTitular(Tarjeta t) { texto.setText(t.getTexto()); }
28     }
29
30     @Override
31     public TarjViewHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
32         View itemView = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_cards, viewGroup, false);
33         itemView.setOnClickListener(this);
34         TarjViewHolder tvh = new TarjViewHolder(itemView);
35         return tvh;
36     }
37
38     @Override
39     public void onBindViewHolder(TarjViewHolder viewHolder, int pos) {
40         Tarjeta item = items.get(pos);
41         viewHolder.bindTitular(item);
42     }
43
44     @Override
45     public int getItemCount() { return items.size(); }
46
47     public void setOnClickListener(View.OnClickListener listener) {
48         this.listener = listener;
49     }
50
51     @Override
52     public void onClick(View view) {
53         if(listener != null)
54             listener.onClick(view);
55     }
56 }
```

En Kotlin:


```

class CardsAdapter(var items: ArrayList<Tarjeta>) : RecyclerView.Adapter<CardsAdapter.TarjViewHolder>() {
    lateinit var onClick: (View) -> Unit1
}
init {
    this.items = items
}
class TarjViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    private var texto: TextView
}
init {
    texto = itemView.findViewById(R.id.txt1)
}
fun bindTarjeta(t: Tarjeta, onClick: (View) -> Unit1) = with(itemView) { this: View
    texto.setText(t.texto)
    setOnClickListener{ onClick(itemView) }
}
}
}
override fun onCreateViewHolder(viewGroup: ViewGroup, viewType: Int): TarjViewHolder {
    val itemView = LayoutInflater.from(viewGroup.context).inflate(R.layout.item_cards, viewGroup, attachToRoot: false)
    return TarjViewHolder(itemView)
}
override fun onBindViewHolder(viewHolder: TarjViewHolder, pos: Int) {
    val item = items.get(pos)
    viewHolder.bindTarjeta(item, onClick)
}
}
override fun getItemCount(): Int {
    return items.size
}
}
}

```

Una vez definido el método, sólo queda llamarlo desde la actividad principal, como se indica:

```

adaptador.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(MainActivity.this, "" + recyclerView.getChildAd
    }
});

```

En Kotlin:

```

adaptador.onClick = {
    Toast.makeText(this@MainActivity, ""+ recyclerView.getChildAdapter
}

```

Notad que el método pertenece al adaptador y no al RecyclerView.

Animación sobre el RecyclerView (ItemAnimator)

El RecyclerView proporciona mediante *ItemAnimator* tres animaciones definidas por defecto que permiten: añadir, eliminar y cambiar ítems del vector de datos.

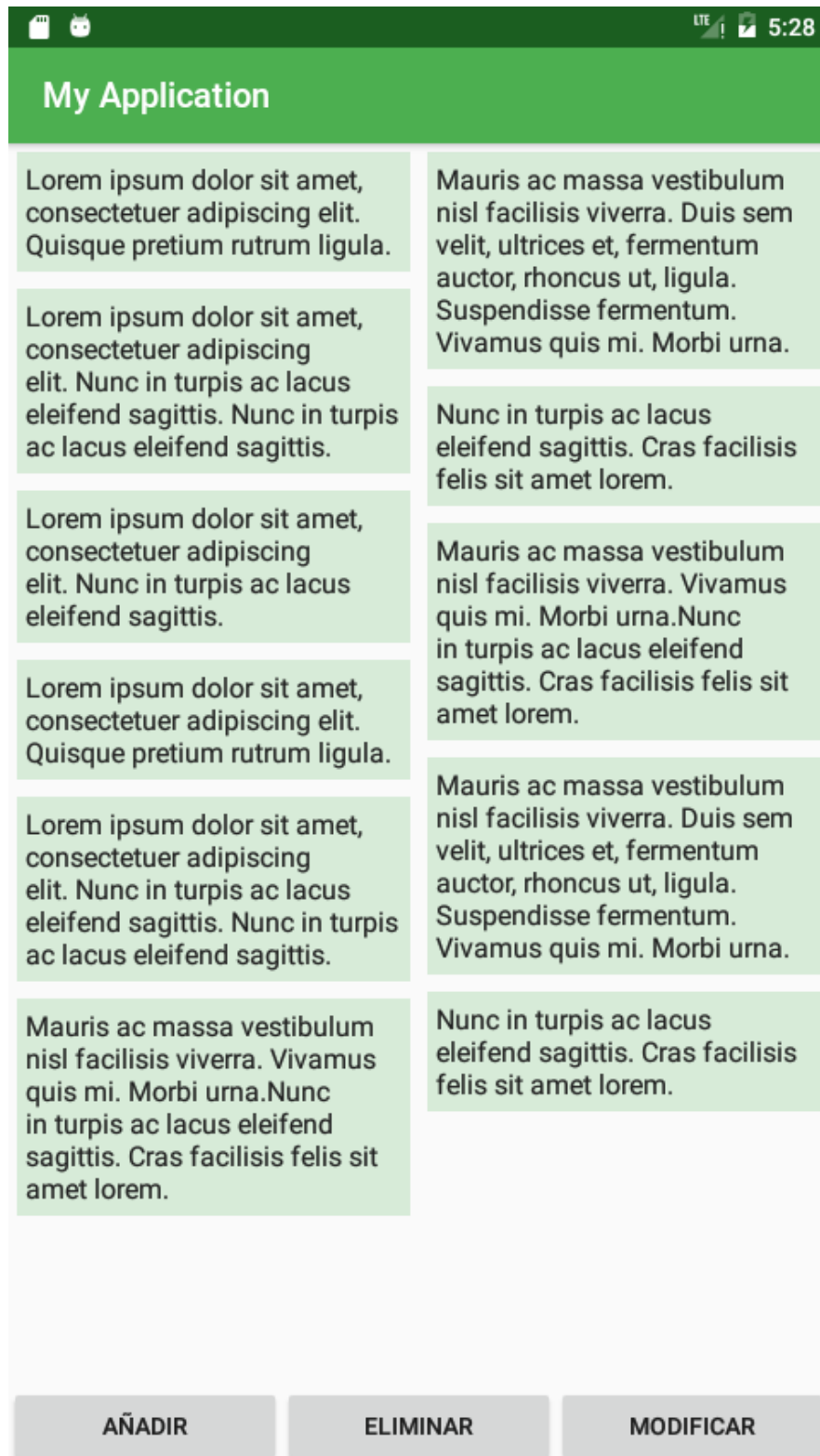
Para ello, definiremos sobre el *RecyclerView*, el uso de *ItemAnimator* como se indica:

```
recView.setItemAnimator(new DefaultItemAnimator());
```

En Kotlin:

```
recView.itemAnimator = DefaultItemAnimator()
```

Para probar las diferentes animaciones, vamos a incluir en el layout principal tres botones, de manera que al pulsar sobre cada uno de ellos se realice la acción correspondiente. El layout quedaría así:



- Añadir un nuevo ítem al vector de datos (en este caso, en la posición 1 del vector).

```
final Button anyadir = (Button) findViewById(R.id.button1);
anyadir.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
```

```

        items.add(1, new Tarjeta(R.string.note7));
        adaptador.notifyItemInserted(1);
    }
});

```

En Kotlin:

```

val anyadir = findViewById(R.id.button1) as Button
anyadir.setOnClickListener(object : View.OnClickListener{
    override fun onClick(v: View?) {
        items.add(1, Tarjeta(R.string.note7))
        adaptador.notifyItemInserted(1)
    }
})

```

- Eliminar un ítem existente del vector de datos (en este caso el ítem que ocupa la posición 1 del vector).

```

final Button eliminar = (Button) findViewById(R.id.button2);
eliminar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        items.remove(1);
        adaptador.notifyItemRemoved(1);
    }
});

```

En Kotlin:

```

val eliminar = findViewById(R.id.button2) as Button
eliminar.setOnClickListener(object : View.OnClickListener{
    override fun onClick(v: View?) {
        items.remove(1)
        adaptador.notifyItemRemoved(1)
    }
})

```

- Cambiar un ítem por otro en el vector de datos (en nuestro caso intercambiamos los ítems de las posiciones 0 y 1).

```
final Button cambiar = (Button) findViewById(R.id.button3);
cambiar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Tarjeta auxiliar = items.get(0);
        items.set(0, items.get(1));
        items.set(1, auxiliar);
        adaptador.notifyDataSetChanged(0,1);
    }
});
```

En Kotlin:

```
val cambiar = findViewById(R.id.button3) as Button
cambiar.setOnClickListener(object : View.OnClickListener{
    override fun onClick(v: View?) {
        val auxiliar = items[0]
        items[0] = items[1]
        items[1] = auxiliar
        adaptador.notifyDataSetChanged(0, 1)
    }
})
```

1.7 Componentes definidos por el usuario

En algunas ocasiones nos puede resultar interesante definir nuestros propios componentes.

Hay tres posibles formas de hacerlo:

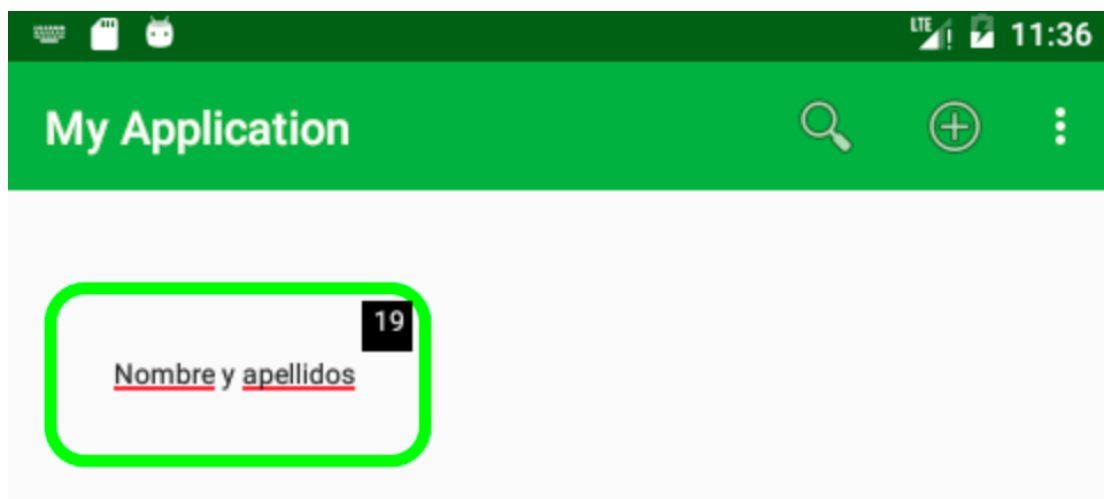
1. Reescribiendo un componente ya existente, para que se adapte a nuestras necesidades.
2. Creando un nuevo componente que estará formado por la agrupación de varios componentes ya existentes.
3. Creando un nuevo componente desde cero.

Vamos a ver un ejemplo de los dos primeros casos, ya que el tercero excede en complejidad para este curso.

Caso 1. Ejemplo EditText Extendido

Vamos a modificar el componente EditText para darle una nueva imagen y funcionalidad. Queremos crear un EditText que presente en la esquina superior derecha, un pequeño recuadro donde nos vaya indicando el número de caracteres que se escriben en el EditText.

El resultado a obtener será el siguiente:



Para ello, realizaremos los siguientes pasos:

1. Escribimos una nueva clase **EditTextExtendido** (EditTextExtendido.java) que

extienda del componente EditText ya existente (AppCompatEditText). El código sería el siguiente:

```
public class EditTextExtendido extends AppCompatEditText {

    private Paint pNegro;
    private Paint pBlanco;

    //funcionamiento en distintas densidades de pantalla
    //escala = getResources().getDisplayMetrics().density;

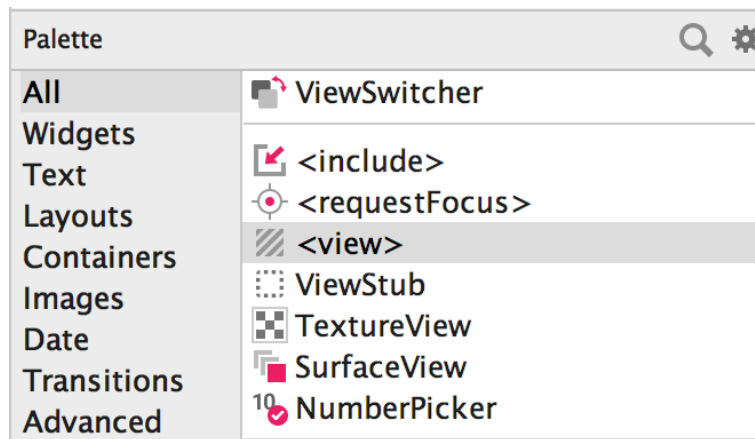
    // Hay que reescribir siempre los constructores heredados.
    // En este caso son tres

    public EditTextExtendido(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        inicializaPinceles();
    }
    public EditTextExtendido(Context context, AttributeSet attrs) {
        super(context, attrs);
        inicializaPinceles();
    }
    public EditTextExtendido(Context context) {
        super(context);
        inicializaPinceles();
    }

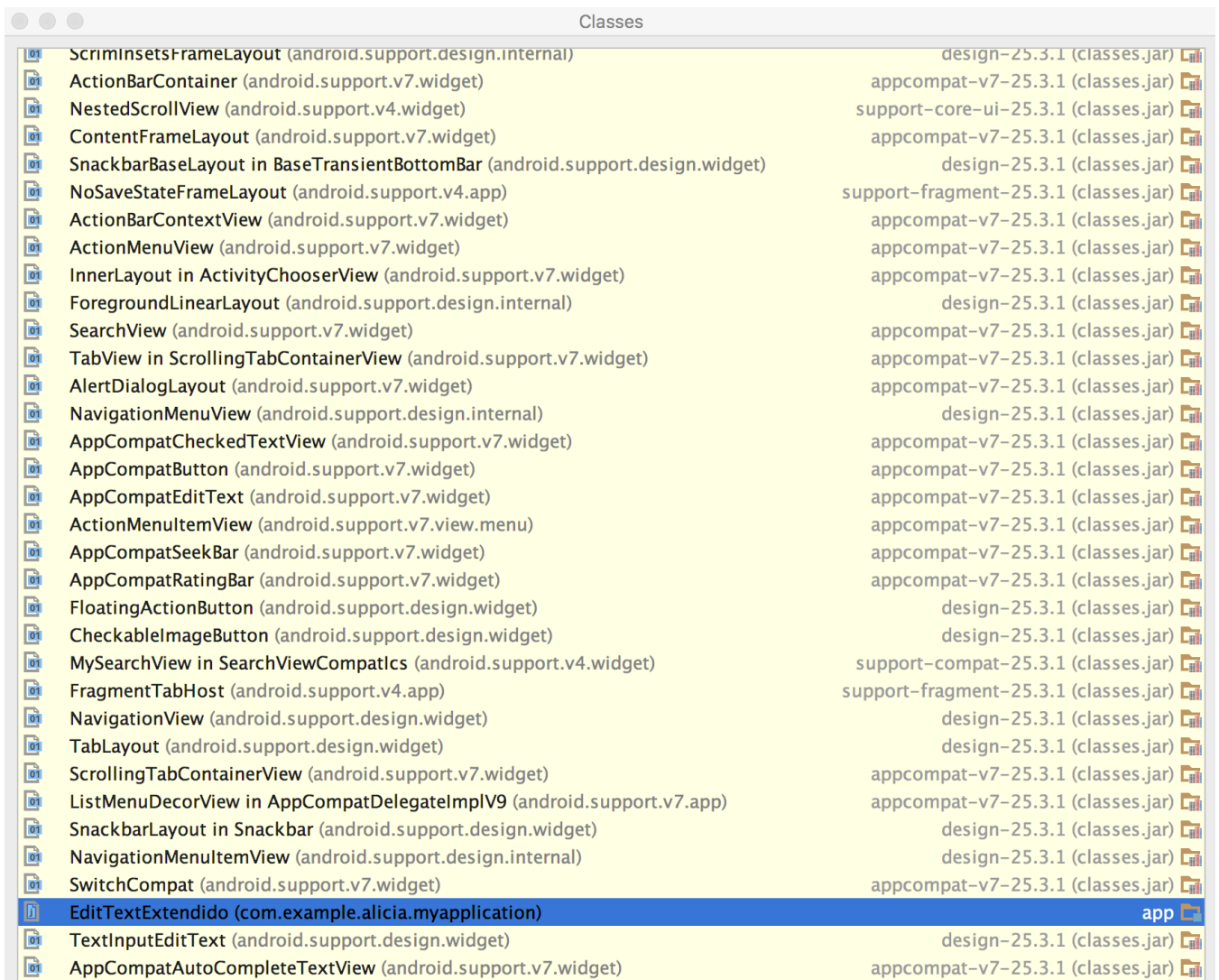
    private void inicializaPinceles() {
        pNegro = new Paint(Paint.ANTI_ALIAS_FLAG);
        pNegro.setColor(Color.BLACK);
        pNegro.setStyle(Style.FILL);
        pBlanco = new Paint(Paint.ANTI_ALIAS_FLAG);
        pBlanco.setColor(Color.WHITE);
    }

    // Para modificar el aspecto del EditText hay que reescribir este método
    @Override
    public void onDraw(Canvas canvas) {
        //Invocamos al método de la superclase (EditText)
        super.onDraw(canvas);
        //Dibujamos el fondo negro del contador en la parte de arriba derecha
        canvas.drawRect(this.getWidth()-30,8,this.getWidth()-8,30, pNegro);
        // Dibujamos el número de caracteres sobre el contador
        canvas.drawText(""+this.getText().toString().length(),this.getWidth()-25,2
    }
}
```

2. A continuación, insertamos en el layout el nuevo componente. En la paleta buscamos el control **view**:



Se despliegan las diferentes clases que podemos asociar, elegimos EditTextExtendido:



El código obtenido quedaría así:


```
<view
    class="com.example.alicia.myapplication.EditTextExtendido"
    id="@+id/view"
    layout_width="match_parent"
    android:layout_width="wrap_content"
    android:layout_height="80dp"
    android:layout_marginTop="24dp"
    android:textSize="12sp" />
```

3. Por último, si quisiéramos darle a nuestro EditText un aspecto lo más parecido posible al ejemplo, tendríamos que definir en la carpeta **drawable**, un fichero donde especificaremos la forma del nuevo componente con borde (stroke) y esquinas redondeadas (corners).

El fichero *rectangulo_redondeado.xml* quedaría como sigue:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <stroke
        android:width="5dp"
        android:color="#00FF00"/>
    <corners
        android:radius="15dp" />
    <padding
        android:left="30dp"
        android:top="30dp"
        android:right="30dp"
        android:bottom="30dp" />
</shape>
```

4. Finalmente, aplicamos sobre el fondo del EditTextExtendido el drawable diseñado:

```
<view
    class="com.example.alicia.myapplication.EditTextExtendido"
    id="@+id/view"
    layout_width="match_parent"
    android:layout_width="wrap_content"
```

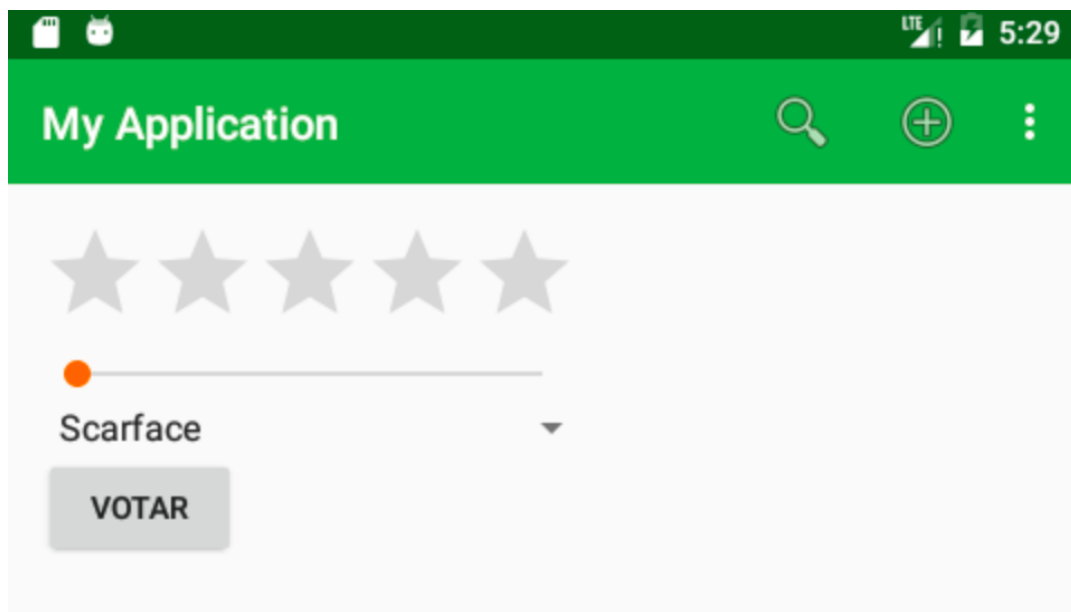
```

android:layout_height="80dp"
android:layout_marginTop="24dp"
android:textSize="12sp"
android:background="@drawable/rectangulo_redondeado"/>

```

Caso 2. Ejemplo Componente Compuesto

Vamos a crear un nuevo componente agrupando varios controles ya existentes. Este nuevo control que será tratado como un todo, estará formado por una RatingBar, una SeekBar, un Spinner y un Button, y tendrá el siguiente aspecto:



1. Primeramente, crearemos un nuevo layout (*compost_voto.xml*) que contenga todos estos componentes:

```

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

    <RatingBar
        android:id="@+id/estrellas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <SeekBar
        android:id="@+id/barra"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

```

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

```
<Button
    android:id="@+id/votar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Votar" />
```

```
</LinearLayout>
```

2. A continuación, crearemos una clase (*ControladorVoto.java*) que extienda de *LinearLayout* como se indica:

```
public class ControladorVoto extends LinearLayout {

    private Button votar;
    private RatingBar estrellas;
    private SeekBar barra;
    private Spinner list_peliculas;

    final String[] _datos = new String[]{"Scarface", "Titanic", "Pretty Woman",

    public ControladorVoto(Context context){
        super(context);
        init(context);
    }

    public ControladorVoto(Context context, AttributeSet attrs){
        super(context, attrs);
        init(context);
    }

    private void init(Context context){
        String infService = Context.LAYOUT_INFLATER_SERVICE;
        LayoutInflater mi_li = (LayoutInflater)getContext().getSystemService(inf

        mi_li.inflate(R.layout.compost_voto, this, true);
        votar = (Button)findViewById(R.id.votar);
        estrellas = (RatingBar)findViewById(R.id.estrellas);
        barra = (SeekBar)findViewById(R.id.barra);
        list_peliculas= (Spinner)findViewById(R.id.list_peliculas);

        barra.setMax(100);

        ArrayAdapter<String> adaptador = new ArrayAdapter<String> (context, andr
```

```
        list_peliculas.setAdapter(adaptador);  
    }  
}
```

3. Con la clase ya definida, insertamos en el layout principal *activity_main.xml*, el nuevo componente con el control view, tal y como hemos hecho en el ejemplo anterior:

```
<view  
    class="com.example.alicia.myapplication.ControladorVoto"  
    id="@+id/view"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

El nuevo componente podrá ser utilizado, tantas veces como deseemos en nuestra aplicación.



