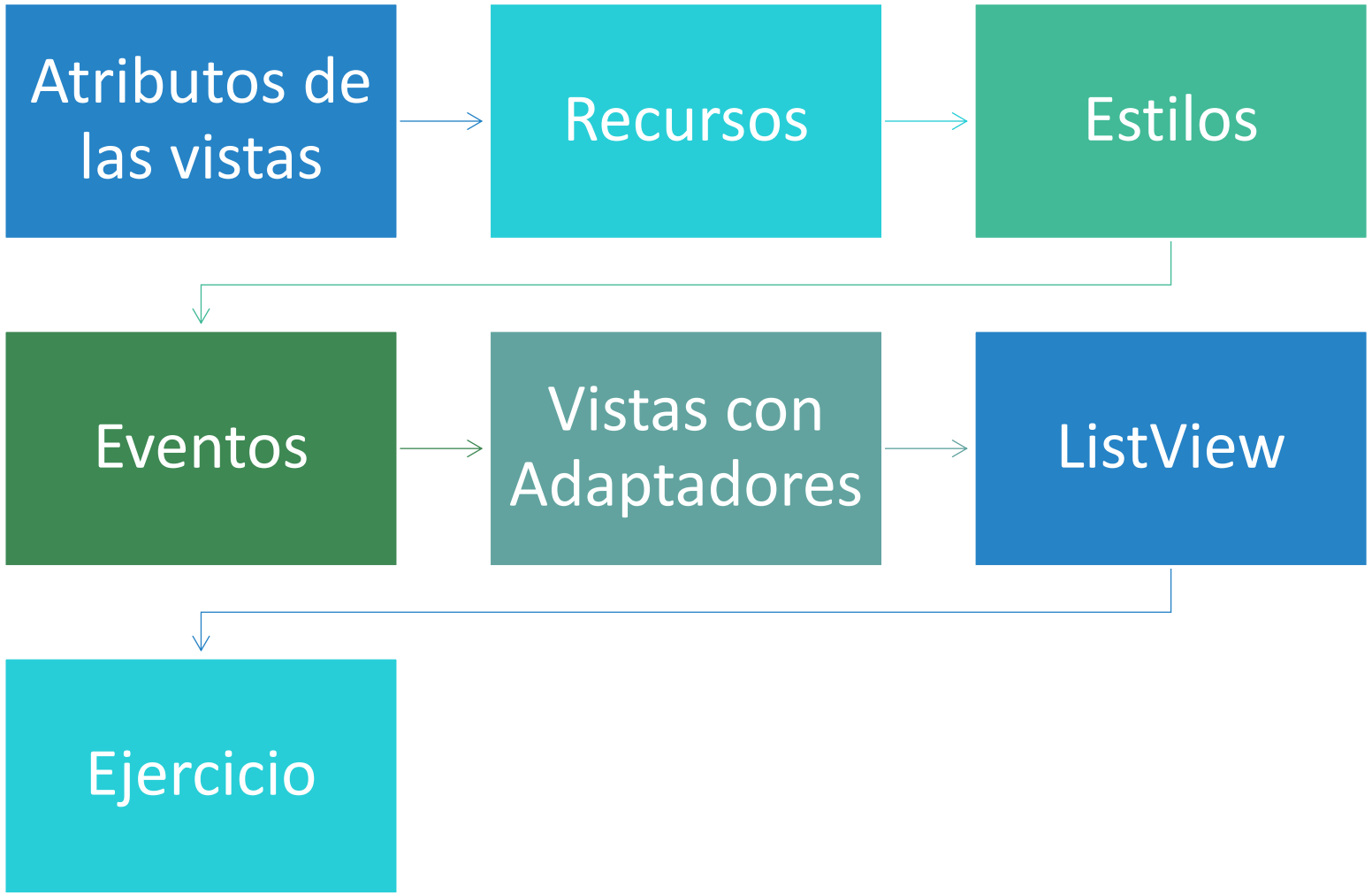
2º DAM

Programación Multimedia y Dispositivos Móviles

Programación Android

José A. Lara

Contenido



android

Atributos de las vistas

Atributos de las vistas

Las propiedades de los Views son las características de cada uno de los elementos. Dentro de estas características podemos encontrar algunas que son útiles para algunos Views pero no son útiles para otros y otras características que serán comunes e importantes para los otros.

Propiedades básicas:

- IDs – El identificador con el que haremos referencia a dicha vista o elemento.
- Dimensiones – El tamaño que tendrá el elemento dentro de la actividad.
- Posiciones – Dónde estará colocado el elemento y respecto a qué lo estará.

No vamos a entrar en todas las posibilidades de los elementos a las que podemos acceder desde aquí porque son muchas, sin embargo, sí que vamos a ver las principales que tendremos que tener en cuenta prácticamente siempre y, según vayamos viendo la necesidad de modificar otras, lo iremos indicando y completando en esta categoría.

Atributos de las vistas

Propiedades

- Layout:width - Anchura del elemento en cuestión.
- Layout:height - Altura del elemento en cuestión.
- Layout:margin - Margen exterior del elemento respecto a los demás.
- Background - Aspectos relacionados con el fondo del elemento.
- Clickable - Indica si el elemento se puede pulsar.
- ID - El identificador del elemento.
- LinearLayout - Dibuja una línea alrededor del View.
- LongClickable - Si la pulsación larga tiene alguna finalidad.
- OnClick - Qué hace cuando se le pulsa.

Atributos de las vistas

Propiedades

- Padding - Margen interior del elemento.
- Rotation - Nos permita rotar el elemento en la pantalla.
- Scale - Configuramos la escala.
- Scrollbar - Nos permite configurar la barra de desplazamiento.
- Tag - La etiqueta del elemento, útil para identificarlo más fácilmente en el desarrollo.
- Text - El texto que tiene el elemento.
- Visibility - Indica la visibilidad del elemento en la actividad.

Recursos

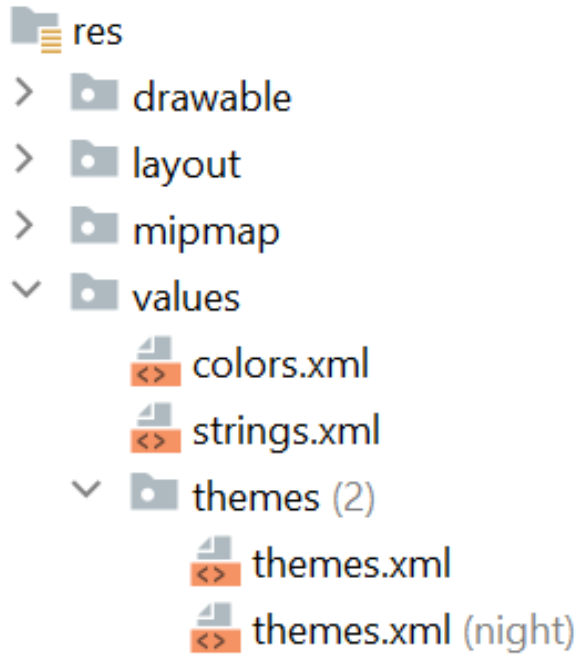
Programación Android

Recursos

Los recursos son los archivos adicionales y el contenido estático que usa tu código, como mapas de bits, definiciones de diseño, strings de interfaz de usuario, instrucciones de animación, etc.

Siempre debes externalizar los recursos para aplicaciones, como imágenes y strings de tu código, para que puedas mantenerlos de forma independiente. También debes proporcionar recursos alternativos para configuraciones de dispositivos específicos, agrupándolos en directorios de recursos con un nombre especial. En tiempo de ejecución, Android utiliza el recurso adecuado según la configuración actual.

Una vez que externalizas los recursos para tu aplicación, puedes acceder a ellos mediante los ID de recursos que se generan en la clase R de tu proyecto. Debes colocar cada tipo de recurso en un subdirectorio específico del directorio res/ de tu proyecto. Por ejemplo, esta es la jerarquía de archivos de un proyecto simple.

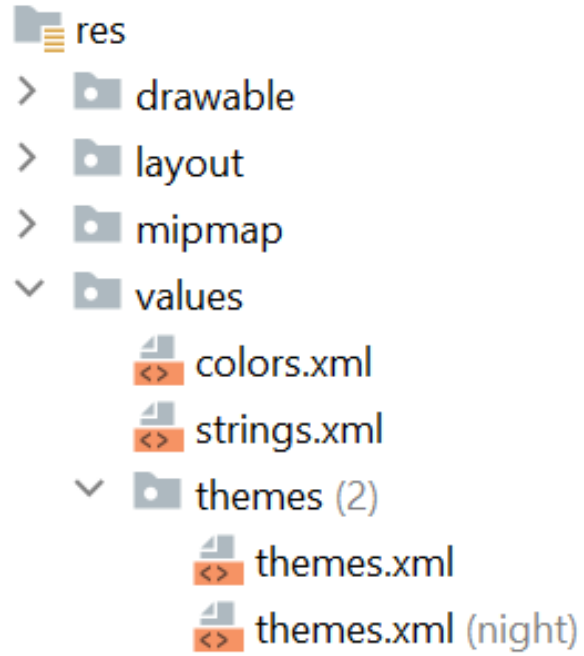


android

Programación Android

Recursos

- drawable/ Normalmente archivos de mapas de bits (.png, .jpg y .gif)
- mipmap/ Archivos de elementos de diseño para diferentes densidades de los íconos de selectores.
- layout/ Archivos XML que definen el diseño de una interfaz de usuario.
- menu/ Archivos XML que definen menús de aplicaciones, como un menú de opciones, un menú contextual o un submenú.
- values/ Archivos XML que contienen valores simples, como strings, valores enteros y colores.
  - arrays.xml para matrices de recursos (matrices escritas).
  - colors.xml para valores de color.
  - dimens.xml para valores de dimensión.
  - strings.xml para valores de strings.
  - styles.xml para estilos.



android

Programación Android

Recursos

Es una práctica muy recomendable utilizar ficheros de recursos para los títulos, etiquetas, mensajes y cualquier otro tipo de elemento textual de la aplicación de forma que facilite el mantenimiento en caso de modificación de los mismos, la reutilización en distintos elementos gráficos, así como la traducción en caso de aplicaciones multi-idioma.

La convención Android determina que el nombre del fichero de cadenas sea strings.xml. Un ejemplo de fichero de cadenas es el siguiente:

```
<resources>
    <string name="app_name">App de recursos humanos</string>
    <string name="titulo">Recursos Humanos Cesur</string>
</resources>
```

android

Recursos

Se puede hacer referencia a los elementos de estos ficheros de forma sencilla desde otros ficheros XML, como por ejemplo desde AndroidManifest.xml o desde los ficheros de Layout para la definición de las View. Para ello, es necesario especificar el elemento siguiendo la notación @tipo/ID, donde tipo es el tipo de recurso requerido e ID es el identificador del mismo:

```
<TextView
    android:id="@+id/tvTexto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="@string/app_name" />
```

Recursos

También se puede hacer referencia a los recursos desde código fuente, a través de una notación similar R.tipo.ID, de forma que desde código fuente se puede disponer de acceso a cualquiera de estos recursos:

Kotlin

```kotlin
val tvTextoPrincipal: TextView = findViewById(R.id.tvTexto)
tvTextoPrincipal.setText(R.string.titulo)
```

Java

```java
TextView tvTextoPrincipal=findViewById(R.id.tvTexto);
tvTextoPrincipal.setText(R.string.app_name);
```

Programación Android

Recursos

Es un caso muy usual el empleo de ficheros de layout desde código fuente para la carga de las distintas View de la aplicación, haciendo referencia directamente al nombre del fichero:

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
```

Estilos

Programación Android

Estilos

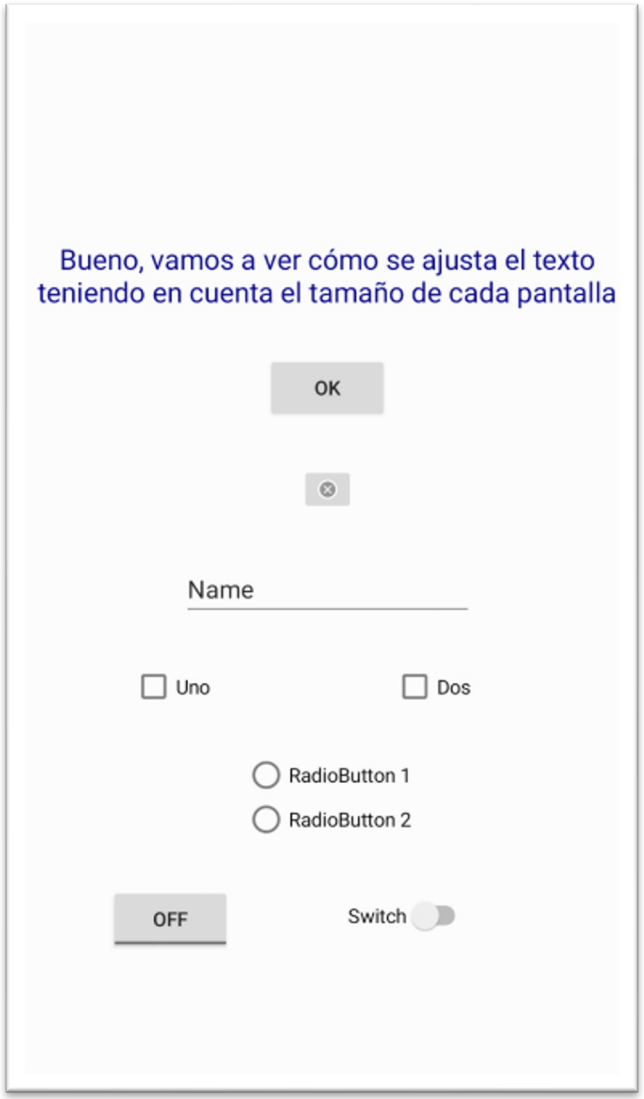
Android permite definir los atributos que definen el aspecto de los elementos visuales con un formato similar al de los estilos CSS. De hecho, es posible crear un fichero exclusivo para la definición de estilos. Para ello, es necesario crear un fichero XML dentro la carpeta /res/values/themes.

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.MyApplication" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

Programación Android

Estilos

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:autoSizeTextType="uniform"
    android:text="@string/Texto_prueba"
    android:textAlignment="center"
    style="@style/CustomText"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.228" />
```



android

Eventos

Programación Android

Eventos

Un evento se produce cuando el usuario interactúa con una aplicación. Los eventos provocan que la aplicación registre de forma automática la ocurrencia de dicha interacción, permitiendo ejecutar acciones concretas en función de los mismos.

Para registrar un evento es necesario establecer un Listener para dicho evento sobre el elemento en concreto para el que se quiere realizar un tratamiento específico. Un Listener representa la clase que se encargará de quedar en espera del evento y actuar en consecuencia.

Programación Android

Eventos

Existen distintos tipos de eventos a los que responder, como los siguientes:

Evento	Descripción
onClick	Al pulsar sobre un elemento
onLongClick	Al mantener pulsado sobre un elemento
onFocusChange	Al cambiar el foco de elemento
onKey	Al pulsar una tecla sobre un elemento
onTouch	Al realizar cualquier evento de pulsación sobre el elemento
onCreateContextMenu	Al crear un menú de contexto para el elemento

Se puede establecer una clase nueva que actúe como Listener o crear un Listener anónimo.

android

Eventos

Una estrategia habitual es emplear como Listener de eventos a la propia clase Activity, ya que desde la misma se tiene acceso al resto de elementos gráficos de la aplicación, de forma que se centraliza toda la gestión de los distintos eventos.

El siguiente ejemplo establece como Listener del evento OnClick a la Activity principal. Para ello es imprescindible que implemente el interfaz OnClickListener, definiendo el comportamiento concreto en el método onClick.

Programación Android

Eventos

```
class MainActivity : AppCompatActivity(), OnClickListener {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val btnAceptar:Button=findViewById(R.id.btnAceptar)
        btnAceptar.setOnClickListener(this)
        val ibCancelar:ImageButton=findViewById(R.id.ibCancelar)
        ibCancelar.setOnClickListener(this)
    }

    override fun onClick(p0: View?) {
        val toast:Toast
        val tvTexto:TextView=findViewById(R.id.tvTexto)
        when (p0?.id){
            R.id.btnAceptar ->{
                toast=Toast.makeText(this.applicationContext, text: "ACEPTAR!!",Toast.LENGTH_SHORT)
                toast.show()
                tvTexto.text="Aceptar"
            }
            R.id.ibCancelar ->{
                toast=Toast.makeText(this.applicationContext, text: "Cancelar", Toast.LENGTH_SHORT)
                toast.show()
                tvTexto.text="Cancelar"
            }
        }
    }
}
```

Programación Android

Eventos

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnAceptar=findViewById(R.id.btnAceptar);
        ImageButton ibCancelar=findViewById(R.id.ibCancelar);
        btnAceptar.setOnClickListener(this);
        ibCancelar.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        Toast toast;
        TextView tvTexto=findViewById(R.id.tvTexto);
        switch (view.getId()){
            case R.id.btnAceptar:
                toast=Toast.makeText(this.getApplicationContext(), text: "ACEPTAR!!",Toast.LENGTH_SHORT);
                tvTexto.setText("Aceptar");
                toast.show();
                break;
            case R.id.ibCancelar:
                toast=Toast.makeText(this.getApplicationContext(), text: "Cancelar",Toast.LENGTH_SHORT);
                tvTexto.setText("Cancelar");
                toast.show();
        }
    }
}
```

android

Eventos

En este caso anterior se centralizan todos los eventos en un mismo método del Activity.

La desventaja de este tipo de implementación es que, al existir un único método, será necesario determinar dentro del mismo cuál es la vista sobre la que recae el evento en cada momento para realizar la acción vinculada a cada una de ellas.

Vistas con adaptadores

Vistas con adaptadores

Para los elementos complejos del interfaz es necesario emplear adaptadores que proporcionen la información necesaria.

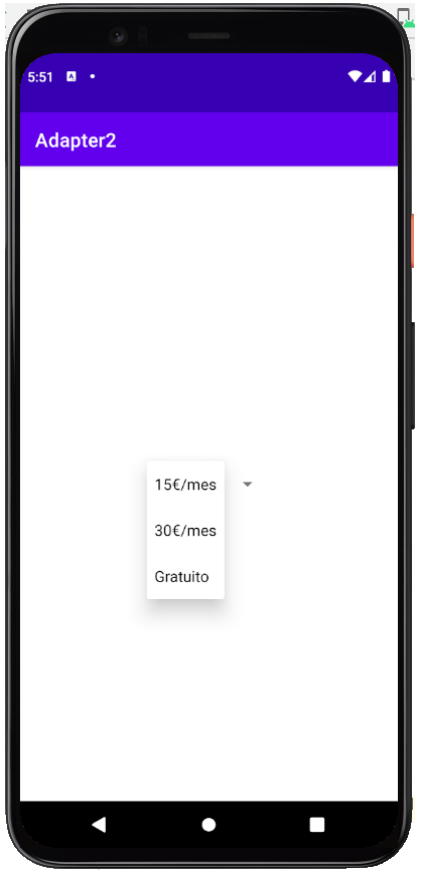
La forma más sencilla de adaptador es emplear un ArrayAdapter relleno con información de un recurso de tipo array. Para poblar un spinner de esta forma en primer lugar es necesario declarar el array en el fichero strings:

```
<resources>
    <string name="app_name">Adapter1</string>
    <string-array name="precios">
        <item>15€/mes</item>
        <item>30€/mes</item>
        <item>Gratuito</item>
    </string-array>
</resources>
```

Programación Android

Vistas con adaptadores

```
class MainActivity : AppCompatActivity(), AdapterView.OnItemSelectedListener {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val sp:Spinner=findViewById(R.id.spinner)
        val adapter=ArrayAdapter.createFromResource( context: this,R.array.precios,
            android.R.layout.simple_spinner_dropdown_item)
        sp.adapter=adapter
    }

    override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
        TODO( reason: "Not yet implemented")
    }

    override fun onNothingSelected(p0: AdapterView<*>?) {
        TODO( reason: "Not yet implemented")
    }
}
```

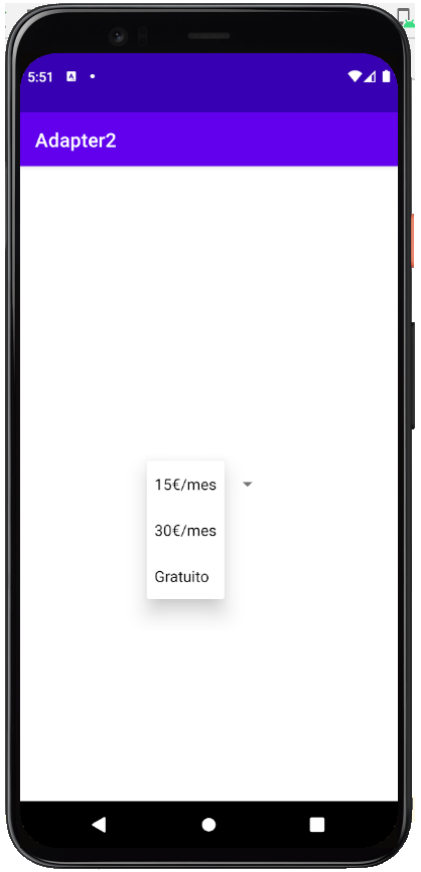


android

Programación Android

Vistas con adaptadores

```
public class MainActivity extends AppCompatActivity implements AdapterView.OnItemSelectedListener{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner sp=findViewById(R.id.spinner);
        sp.setOnItemSelectedListener(this);
        ArrayAdapter<?> adapter=ArrayAdapter.createFromResource( context: this,R.array.precios,
                android.R.layout.simple_spinner_dropdown_item);
        sp.setAdapter(adapter);
    }

    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {

    }

    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {

    }
}
```



android

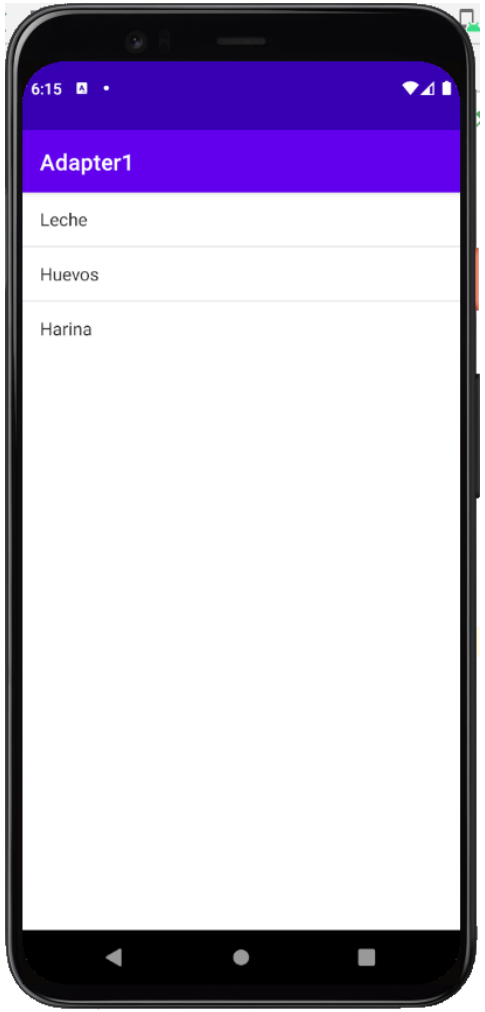
Vistas con adaptadores

De la misma forma se puede completar una lista. Para ello en primer lugar se define un elemento de tipo **ListView**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

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

Programación Android

Vistas con adaptadores



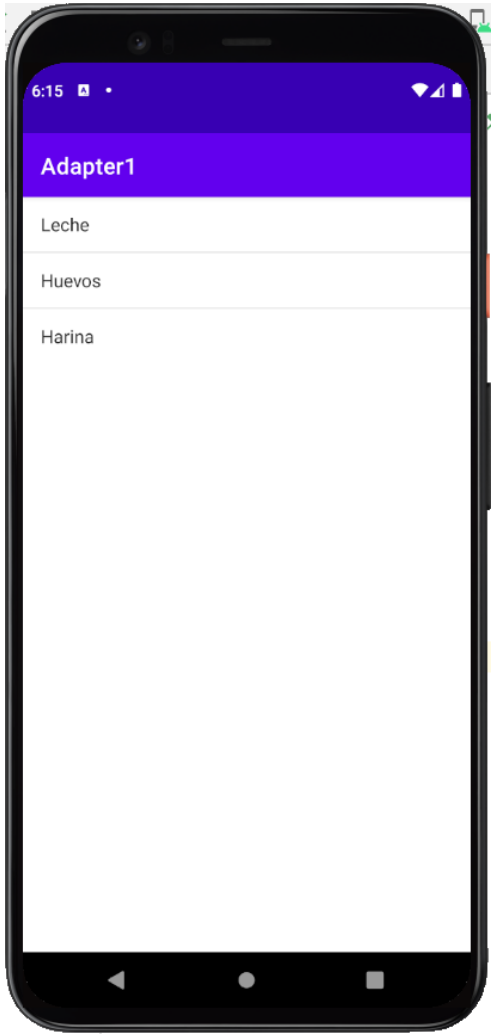

```
class MainActivity : AppCompatActivity(){
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val lv:ListView=findViewById(R.id.lvLista)
        val items=arrayOf("Leche","Huevos","Harina")
        val adapter=ArrayAdapter( context: this,android.R.layout.simple_list_item_1,items)
        lv.adapter=adapter
    }
}
```

android

Programación Android

Vistas con adaptadores

```
public class MainActivity extends AppCompatActivity{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView lv=findViewById(R.id.lvLista);
        String[] items={"Leche","Huevos","Harina"};
        ArrayAdapter<String> adapter=new ArrayAdapter<String>( context: this,
                android.R.layout.simple_list_item_1,items);
        lv.setAdapter(adapter);
    }
}
```



android

Vistas con adaptadores

Tal y como veremos, el caso más completo es emplear un **Adapter** propio que permitirá, no solo mostrar el texto deseado, sino otros elementos como imágenes vinculadas. Además cada elemento estará representado por un objeto completo, de forma que al seleccionar el elemento se puedan realizar acciones más especificas.

ListView

ListView

Un **ListView** es una lista de views. Visualiza una colección de vistas en un scroll vertical, donde cada vista está posicionada inmediatamente después de la vista previa en la lista (o sea, que se visualizan una detrás de otra).

Si queremos un enfoque más moderno, flexible y eficaz en cuanto a la visualización de las listas utilizaremos los **RecyclerView** (que veremos más adelante).

Para visualizar una lista, incluiremos un **ListView** en el archivo XML de nuestro Layout.

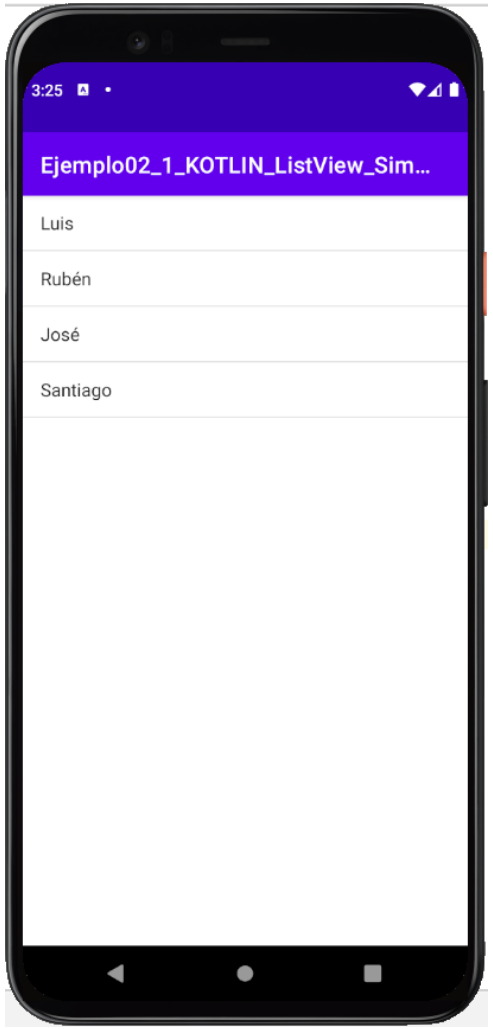
ListView

Un **ListView** es un adaptador de listas que no conoce los detalles de los datos que va a visualizar. Para ello requiere de un adaptador (ListAdapter) que las visualice correctamente y que proporcione un scroll.

Existen adaptadores simples predefinidos en Android, aunque normalmente vamos a construir nuestro propio adaptador.

Programación Android

ListView



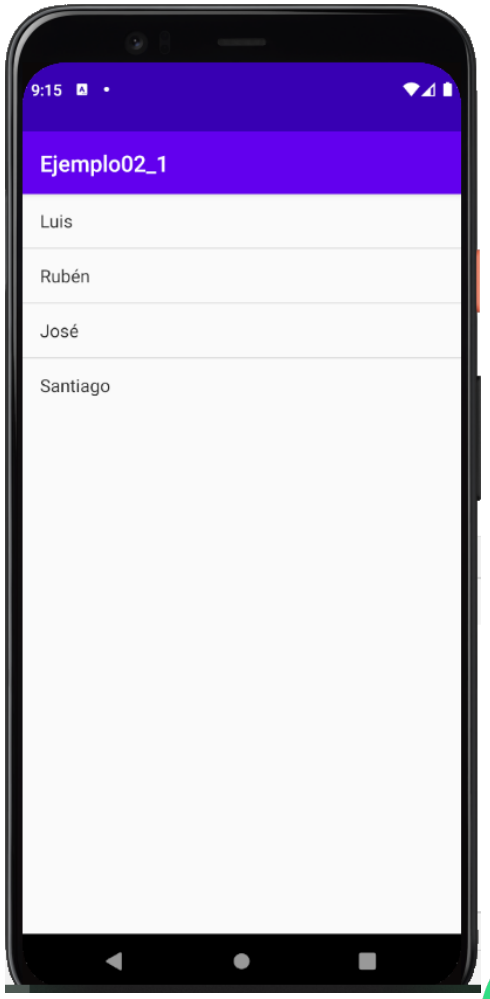

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val lv1: ListView =findViewById(R.id.lvLista)
        val items = arrayOf("Luis", "Rubén", "José", "Santiago")
        val adapter = ArrayAdapter( context: this,
            android.R.layout.simple_list_item_1, items
        )
        lv1.adapter=adapter
    }
}
```

android

Programación Android

ListView

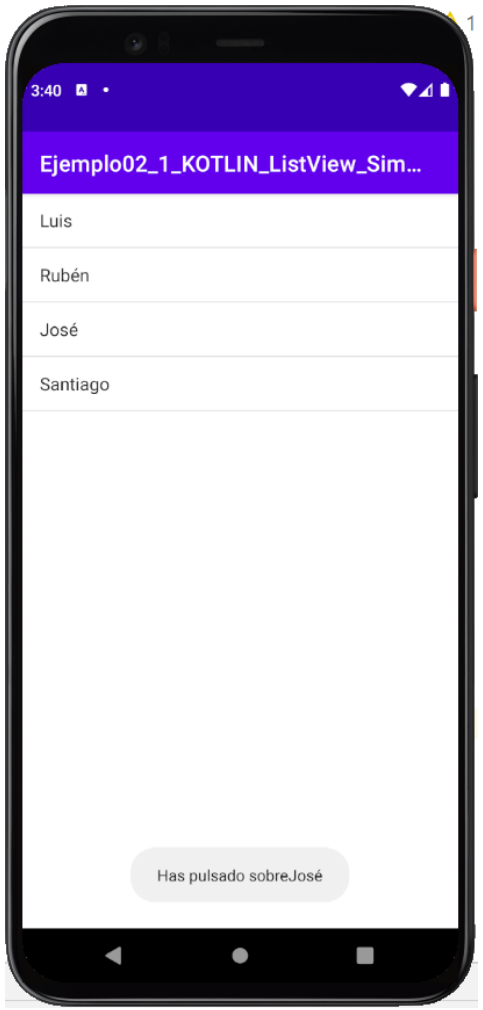
```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView lv1 = findViewById(R.id.lvLista);
        String[] items = {"Luis", "Rubén", "José", "Santiago" };
        ArrayAdapter<String> adapter = new ArrayAdapter<String>( context: this,
                                    android.R.layout.simple_list_item_1,items);
        lv1.setAdapter(adapter);
    }
}
```



android

Programación Android

ListView – Evento OnItemClick

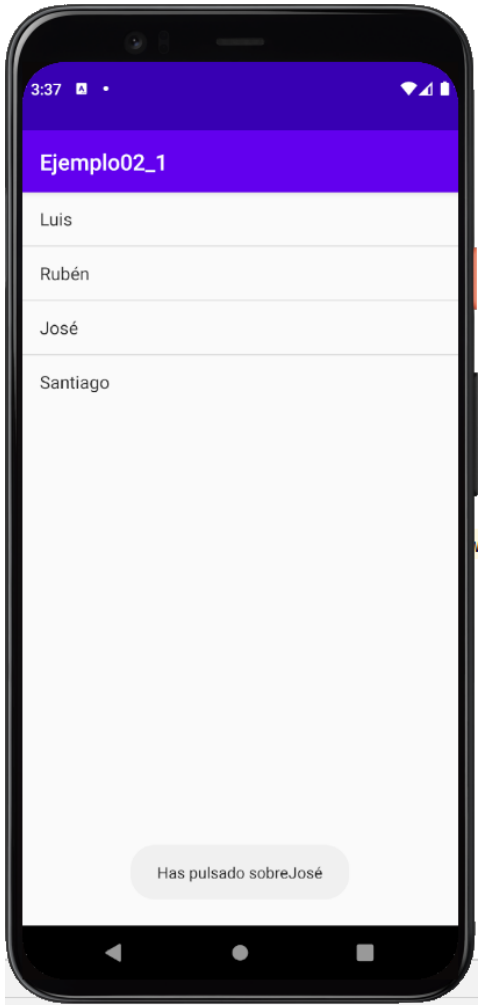



```
lv1.onItemClickListener =
    OnItemClickListener { parent, view, position, id ->
        Toast.makeText(
            context: this@MainActivity,  text: "Has pulsado sobre" + items[position],
            Toast.LENGTH_SHORT
        ).show()
    }
```

android

Programación Android

ListView – Evento OnItemClick




```
lv1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText( context: MainActivity.this, text: "Has pulsado sobre"+items[position],
                Toast.LENGTH_SHORT).show();
    }
});
```

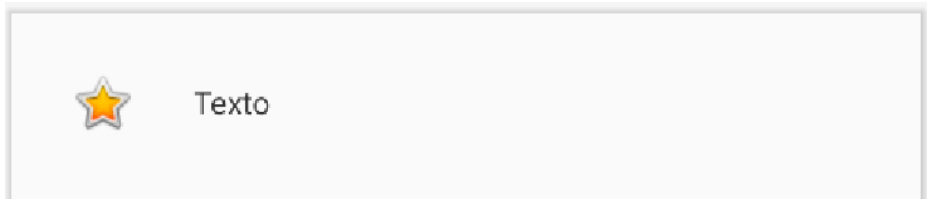
android

Programación Android

Personalizar un ListView

Vamos a ver una personalización básica de un ListView con un Adapter.

En primer lugar tenemos que crear un nuevo Layout para definir las vistas que se van a mostrar en el ListView. Se va a mostrar en la parte izquierda una imagen y en la derecha un texto.




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="80dp">
    <ImageView
        android:id="@+id/ivImagen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_margin="25dp"
        app:srcCompat="@android:drawable/btn_star_big_on" />
    <TextView
        android:id="@+id/tvTexto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="#333333"
        android:layout_gravity="center_vertical"
        android:text="@string/texto" />
</LinearLayout>
```

android

Personalizar un ListView

En segundo lugar creamos una clase **MyAdapter** que hereda de **BaseAdapter**.

```
class MyAdapter: BaseAdapter() {
    override fun getCount(): Int {
        TODO( reason: "Not yet implemented")
    }

    override fun getItem(p0: Int): Any {
        TODO( reason: "Not yet implemented")
    }

    override fun getItemId(p0: Int): Long {
        TODO( reason: "Not yet implemented")
    }

    override fun getView(p0: Int, p1: View?, p2: ViewGroup?): View {
        TODO( reason: "Not yet implemented")
    }
}
```

Personalizar un ListView

En segundo lugar creamos una clase **MyAdapter** que hereda de **BaseAdapter**.

```
public class MyAdapter extends BaseAdapter {
    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        return null;
    }
}
```

Programación Android

ListView

- El método **getCount()** va a proporcionar al adaptador el número de ítems.
- El método **getItem(int position)** nos va a proporcionar un elemento determinado de la colección.
- El método **getItemId(int id)** nos proporciona el id del elemento en esa colección.
- El método **getView(int position, View convertView, ViewGroup viewGroup)** es donde realmente dibuja cada uno de los elementos del ListView.

```
public class MyAdapter extends BaseAdapter {
    private Context context;
    private int layout;
    private List<String> nombres;

    public MyAdapter(Context context, int layout, List<String> nombres) {
        this.context = context;
        this.layout = layout;
        this.nombres = nombres;
    }

    @Override
    public int getCount() { return nombres.size(); }

    @Override
    public Object getItem(int position) { return nombres.get(position); }

    @Override
    public long getItemId(int id) { return id; }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View vista = convertView;
        LayoutInflater layoutInflater=LayoutInflater.from(context);
        vista=layoutInflater.inflate(R.layout.list_item, root: null);
        String nombre=nombres.get(position);
        TextView tv=vista.findViewById(R.id.tvTexto);
        tv.setText(nombre);
        return vista;
    }
}
```

android

Programación Android

ListView

```
class MyAdapter(
    private val context: Context,
    private val layout: Int,
    private val nombres: List<String>
) :
    BaseAdapter() {
    override fun getCount(): Int {
        return nombres.size
    }

    override fun getItem(position: Int): Any {
        return nombres[position]
    }

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

    override fun getView(position: Int, convertView: View, parent: ViewGroup): View {
        var vista = convertView
        val layoutInflater = LayoutInflater.from(context)
        vista = layoutInflater.inflate(R.layout.list_item,  root: null)
        val nombre = nombres[position]
        val tv = vista.findViewById<TextView>(R.id.tvTexto)
        tv.text = nombre
        return vista
    }
}
```

android

Programación Android

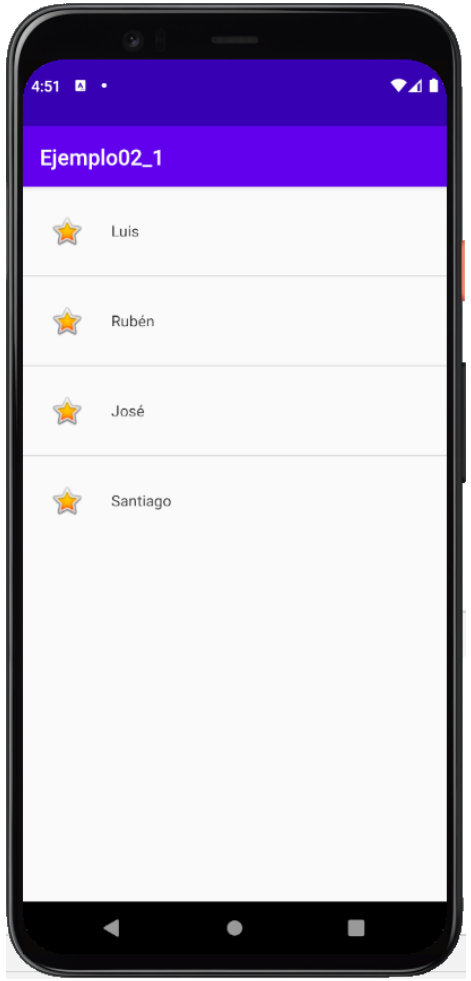
ListView

```
public class MyAdapter extends BaseAdapter {
    private Context context;
    private int layout;
    private List<String> nombres;

    public MyAdapter(Context context, int layout, List<String> nombres) {
        this.context = context;
        this.layout = layout;
        this.nombres = nombres;
    }

    @Override
    public int getCount() { return nombres.size(); }

    @Override
    public Object getItem(int position) { return nombres.get(position); }

    @Override
    public long getItemId(int id) { return id; }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View vista = convertView;
        LayoutInflater layoutInflater=LayoutInflater.from(context);
        vista=layoutInflater.inflate(R.layout.list_item, root: null);
        String nombre=nombres.get(position);
        TextView tv=vista.findViewById(R.id.tvTexto);
        tv.setText(nombre);
        return vista;
    }
}
```

android

Programación Android

ListView

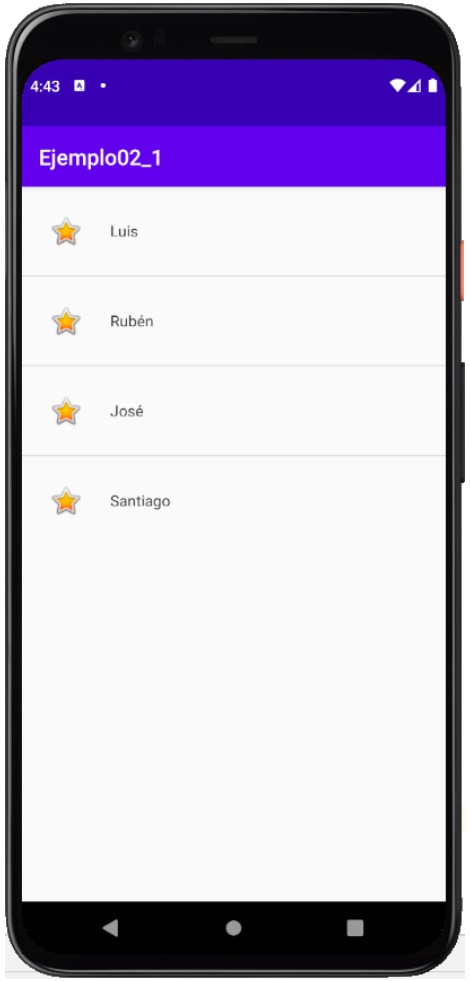



```
val lv1 = findViewById<ListView>(R.id.lvLista)
val items: MutableList<String> = ArrayList()
items.add("Luis")
items.add("Rubén")
items.add("José")
items.add("Santiago")
val adapter = MyAdapter( context: this, R.layout.list_item, items)
lv1.setAdapter(adapter)
lv1.setOnItemClickListener(OnItemClickListener { parent, view, position, id ->
    Toast.makeText(
        context: this@MainActivity,  text: "Has pulsado sobre" + items[position],
        Toast.LENGTH_SHORT
    ).show()
})
```

android

Programación Android

ListView




```
ListView lv1 = findViewById(R.id.lvLista);
List<String> items=new ArrayList<String>();
items.add("Luis");
items.add("Rubén");
items.add("José");
items.add("Santiago");
MyAdapter adapter=new MyAdapter( context: this,R.layout.list_item,items);
lv1.setAdapter(adapter);
lv1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText( context: MainActivity.this, text: "Has pulsado sobre"+items.get(position),
                Toast.LENGTH_SHORT).show();
    }
});
```

android

Personalizar un ListView

Android solo renderiza los elementos que **están en pantalla**. Cada vez que va a cargar un elemento ejecuta el método **getView**, y tenemos llamadas al método **findViewById**, que es una llamada "pesada" ya que el archivo R es un archivo muy grande. En nuestro ejemplo, localiza el layout en el inflater y después el TextView en el findViewById.

```
override fun getView(position: Int, convertView: View, parent: ViewGroup): View {
    var vista = convertView
    val layoutInflater = LayoutInflater.from(context)
    vista = layoutInflater.inflate(R.layout.list_item,  root: null)
    val nombre = nombres[position]
    val tv = vista.findViewById<TextView>(R.id.tvTexto)
    tv.text = nombre
    return vista
}
```

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    View vista = convertView;
    LayoutInflater layoutInflater=LayoutInflater.from(context);
    vista=layoutInflater.inflate(R.layout.list_item, root: null);
    String nombre=nombres.get(position);
    TextView tv=vista.findViewById(R.id.tvTexto);
    tv.setText(nombre);
    return vista;
}
```

Personalizar un ListView

Para evitar esto utilizaremos un **patrón** (**ViewHolder pattern**), de forma que aumentaremos la productividad ya que no tendrá que hacer para cada elemento esas llamadas.

En primer lugar crearemos una **clase interna** (Kotlin) o una **clase static** (Java) dentro de nuestra clase MyAdapter, que llamaremos **ViewHolder**.

En segundo lugar **modificamos** el método **getView()**.

Personalizar un ListView

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View? {
    var convertView = convertView
    val holder: ViewHolder
    if (convertView == null) {
        val layoutInflater = LayoutInflater.from(context)
        convertView = layoutInflater.inflate(R.layout.list_item,  root: null)
        holder = ViewHolder()
        holder.texto = convertView.findViewById(R.id.tvTexto)
        convertView.tag = holder
    } else {
        holder = convertView.tag as ViewHolder
    }
    val nombre = nombres[position]
    holder.texto!!.text = nombre
    return convertView
}

internal class ViewHolder {
    var foto: ImageView? = null
    var texto: TextView? = null
}
```

android

Programación Android

Personalizar un ListView

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;
    if (convertView==null) {
        LayoutInflater layoutInflater = LayoutInflater.from(context);
        convertView = layoutInflater.inflate(R.layout.list_item,  root: null);
        holder = new ViewHolder();
        holder.texto = convertView.findViewById(R.id.tvTexto);
        convertView.setTag(holder);
    } else {
        holder=(ViewHolder) convertView.getTag();
    }
    String nombre= nombres.get(position);
    holder.texto.setText(nombre);
    return convertView;
}
static class ViewHolder {
    ImageView foto;
    TextView texto;
}
```

android

Ejercicio

Programación Android

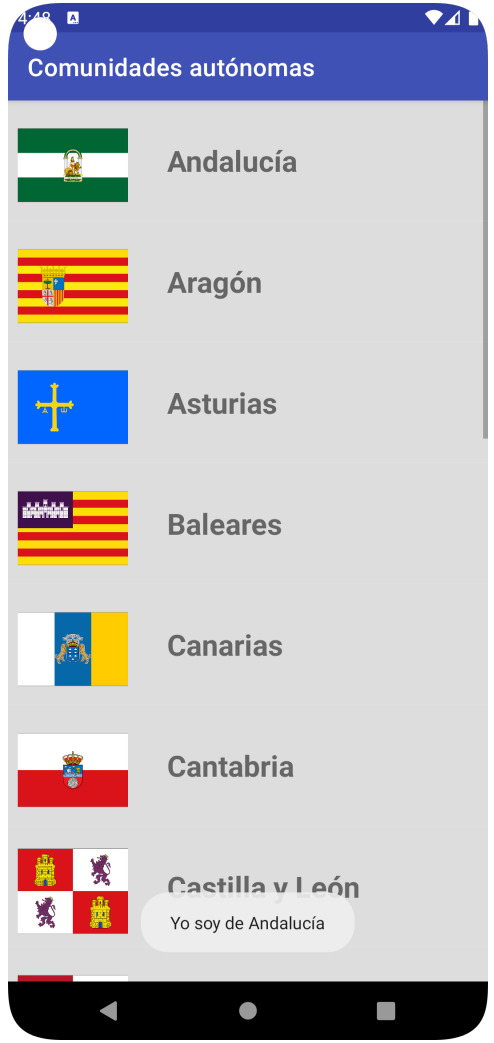
Ejercicio

Crear una aplicación Android que muestre una lista con las comunidades y ciudades autónomas de España. Para ello debéis emplear un **ListView**.

Cuando un usuario seleccione alguno de los elementos debe aparecer un Toast con el mensaje:

"Yo soy de Andalucía"

(o de Extremadura, Canarias… según la comunidad seleccionada)



android