# Interfaz de usuario en Android: Controles de selección (II)

En el artículo anterior ya comenzamos a hablar de los controles de selección en Android, empezando por explicar el concepto de *adaptador* y describiendo el control Spinner. En este nuevo artículo nos vamos a centrar en el control de selección más utilizado de todos, el `ListView`.

Un control `ListView` muestra al usuario una lista de opciones seleccionables directamente sobre el propio control, sin listas emergentes como en el caso del control`Spinner`. En caso de existir más opciones de las que se pueden mostrar sobre el control se podrá por supuesto hacer *scroll* sobre la lista para acceder al resto de elementos.

Para empezar, veamos cómo podemos añadir un control `ListView` a nuestra interfaz de usuario:

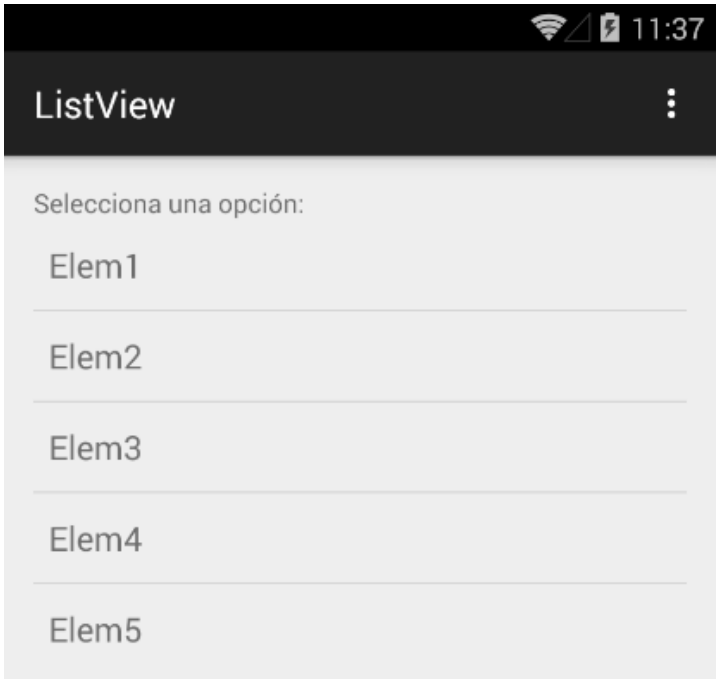
```
<ListView android:id="@+id/LstOpciones"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Una vez más, podremos modificar el aspecto del control utilizando las propiedades de fuente y color ya comentadas en artículos anteriores. Por su parte, para enlazar los datos con el control podemos utlizar por ejemplo el mismo código que ya vimos para el control `Spinner`. Definiremos primero un array con nuestros datos de prueba, crearemos posteriormente el adaptador de tipo `ArrayAdapter` y lo asignaremos finalmente al control mediante el método `setAdapter()`:

```
final String[] datos =
new String[]{"Elem1","Elem2","Elem3","Elem4","Elem5"};

ArrayAdapter<String> adaptador =
new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, datos);

lstOpciones = (ListView)findViewById(R.id.LstOpciones);

lstOpciones.setAdapter(adaptador);
```

**NOTA:** En caso de necesitar mostrar en la lista datos procedentes de una base de datos la mejor práctica es utilizar un `Loader` (concretamente un CursorLoader), que cargará los datos de forma asíncrona de forma que la aplicación no se bloquee durante la carga. Esto lo veremos más adelante en el curso, ahora nos conformaremos con cargar datos estáticos procedentes de un array.

En el código anterior, para mostrar los datos de cada elemento hemos utilizado otro layout genérico de Android para los controles de tipo `ListView`(`android.R.layout.simple_list_item_1`), formado únicamente por un `TextView` con unas dimensiones determinadas.




Como podéis comprobar el uso básico del control `ListView` es completamente análogo al ya comentado para el control `Spinner`.

Hasta aquí todo sencillo. Pero, ¿y si necesitamos mostrar datos más complejos en la lista? ¿Qué ocurre si necesitamos que cada elemento de la lista esté

formado a su vez por varios elementos? Pues vamos a provechar este artículo dedicado a los `ListView` para ver cómo podríamos conseguirlo, aunque todo lo que comentaré es extensible a otros controles de selección.

Para no complicar mucho el tema vamos a hacer que cada elemento de la lista muestre por ejemplo dos líneas de texto a modo de título y subtítulo con formatos diferentes (por supuesto se podrían añadir muchos más elementos, por ejemplo imágenes, *checkboxes*, etc).

En primer lugar vamos a crear una nueva clase java para contener nuestros datos de prueba. Vamos a llamarla `Titular` y tan sólo va a contener dos atributos, título y subtítulo.

```
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;
   }
}
```

En cada elemento de la lista queremos mostrar ambos datos, por lo que el siguiente paso será crear un layout XML con la estructura que deseemos. En mi caso voy a mostrarlos en dos etiquetas de texto (`TextView`), la primera de ellas en negrita y con un tamaño de letra un poco mayor. Llamaremos a este layout "*listitem_titular.xml*":

```
<?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">
```

```
    <TextView android:id="@+id/LblTitulo"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:textStyle="bold"
       android:textSize="20sp" />

    <TextView android:id="@+id/LblSubTitulo"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:textStyle="normal"
       android:textSize="12sp" />

</LinearLayout>
```

Ahora que ya tenemos creados tanto el soporte para nuestros datos como el layout que necesitamos para visualizarlos, lo siguiente que debemos hacer será indicarle al adaptador cómo debe utilizar ambas cosas para generar nuestra interfaz de usuario final. Para ello vamos a crear nuestro propio adaptador extendiendo de la clase `ArrayAdapter`.

```
class AdaptadorTitulares extends ArrayAdapter<Titular> {


    public AdaptadorTitulares(Context context, Titular[] datos) {
        super(context, R.layout.listitem_titular, datos);
    }

public View getView(int position, View convertView, ViewGroup parent)
{
    LayoutInflater inflater = LayoutInflater.from(getContext());
    View itemView = inflater.inflate(R.layout.listitem_titular, null);

    TextView lblTitulo =
(TextView)itemView.findViewById(R.id.LblTitulo);
    lblTitulo.setText(getItem(position).getTitulo());

    TextView lblSubtitulo =
(TextView)itemView.findViewById(R.id.LblSubTitulo);
    lblSubtitulo.setText(getItem(position).getSubtitulo());

    return(itemView);
}

}
```

Analicemos el código anterior. Lo primero que encontramos es el constructor para nuestro adaptador, al que sólo pasaremos el *contexto* (que normalmente será la actividad desde la que se crea el adaptador) y el array de datos a mostrar, que en nuestro caso es un array de objetos de tipo `Titular`. En este constructor tan sólo llamaremos al constructor padre tal como ya vimos al principio de este artículo, pasándole nuestros dos parámetros (contexto y

datos) y el ID del layout que queremos utilizar (en nuestro caso el nuevo que hemos creado, `listitem_titular`).

Posteriormente, redefinimos el método encargado de generar y rellenar con nuestros datos todos los controles necesarios de la interfaz gráfica de cada elemento de la lista. Este método es `getView()`.

El método `getView()` se llamará cada vez que haya que mostrar un elemento de la lista. Lo primero que debe hacer es "*inflar*" el layout XML que hemos creado. Esto consiste en consultar el XML de nuestro layout y crear e inicializar la estructura de objetos java equivalente. Para ello, crearemos un nuevo objeto `LayoutInflater` y generaremos la estructura de objetos mediante su método `inflate(id_layout)`.

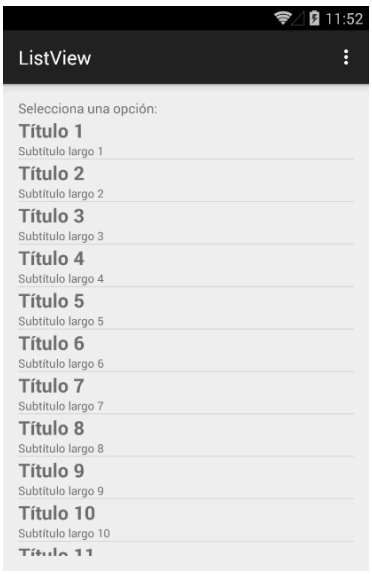
Tras esto, tan sólo tendremos que obtener la referencia a cada una de nuestras etiquetas como siempre lo hemos hecho y asignar su texto correspondiente según los datos de nuestro array y la posición del elemento actual (parámetro `position` del método `getView()`). Para obtener el elemento de nuestro Array en dicha posición utilizamos el método heredado getItem(int posición)

Una vez tenemos definido el comportamiento de nuestro adaptador la forma de proceder en la actividad principal será análoga a lo ya comentado, definiremos el array de datos de prueba, crearemos el adaptador y lo asignaremos al control mediante `setAdapter()`:

```
new Titular[]{
        new Titular("Título 1", "Subtítulo largo 1"),
        new Titular("Título 2", "Subtítulo largo 2"),
        ......
        new Titular("Título 15", "Subtítulo largo 15")};
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
lstOpciones = (ListView)findViewById(R.id.LstOpciones);
//Adaptador
AdaptadorTitulares adaptador =
```

```
        new AdaptadorTitulares(this, datos);

lstOpciones.setAdapter(adaptador);
```

Hecho esto, y si todo ha ido bien, nuestra nueva lista debería quedar como vemos en la imagen siguiente:




Otra posible personalización de nuestra lista podría ser **añadirle una cabecera o un pie**. Para esto, definiremos un nuevo layout XML para la cabecera/pie y lo añadiremos a la lista antes de asignar el adaptador.

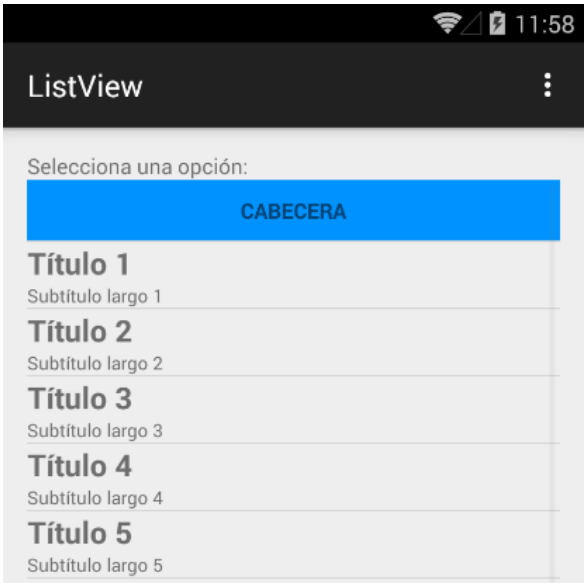
Así por ejemplo, podríamos crear la siguiente cabecera compuesta por una etiqueta de texto centrada y en negrita sobre fondo azul, en un fichero XML situado en *layout/list_header.xml*:

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:text="CABECERA"
        android:textStyle="bold"
        android:background="#ff0093ff"
        android:gravity="center" />

</LinearLayout>
```

Y una vez creada la añadiríamos a nuestra lista inflando su layout y llamando al método **`addHeaderView()`** con la vista resultante:

```
lstOpciones = (ListView)findViewById(R.id.LstOpciones);
//Cabecera
View header = getLayoutInflater().inflate(R.layout.list_header, null);
lstOpciones.addHeaderView(header);
```

Éste sería un ejemplo sencillo, pero tanto las cabeceras como el pie de lista pueden contener por supuesto otros elementos como imágenes o botones. Veamos cómo quedaría:




Por último comentemos un poco los eventos de este tipo de controles. Si quisiéramos realizar cualquier acción al pulsarse sobre un elemento de la lista creada tendremos que implementar el evento **`onItemClick`**. Veamos cómo con un ejemplo:

```
//Eventos
lstOpciones.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> a, View v, int position,
long id) {

        //Alternativa 1:
        String opcionSeleccionada =
```

```
                ((Titular)a.getItemAtPosition(position)).getTitulo();

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

        lblEtiqueta.setText("Opción seleccionada: " +
opcionSeleccionada);
    }
});
```

Este evento recibe 4 parámetros:

- Referencia al control lista que ha recibido el click (`AdapterView a`).
- Referencia al objeto `View` correspondiente al ítem pulsado de la lista (`View v`).
- Posición del elemento pulsado dentro del adaptador de la lista (`int position`).
- Id del elemento pulsado (`long id`).

Con todos estos datos, si quisiéramos por ejemplo mostrar el título de la opción pulsada en la etiqueta de texto superior (`lblEtiqueta`) tendríamos dos posibilidades:

1. Acceder a la vista asociada al adaptador y a partir de ésta obtener mediante `getItemAtPosition()` el elemento cuya posición sea position. Esto nos devolvería un objeto de tipo `Titular`, por lo que obtendríamos el título llamando a su método `getTitulo()`.
2. Acceder directamente a la vista que se ha pulsado, que tendría la estructura definida en nuestro layout personalizado `listitem_titular.xml`, y obtener mediante `findViewById()` y `getText()` el texto del control que alberga el campo título.

Y esto sería todo por ahora. Aunque ya sabemos utilizar y personalizar las listas en Android, en el próximo apartado daremos algunas indicaciones para

utilizar de una forma mucho más eficiente los controles de este tipo, algo que los usuarios de nuestra aplicación agradecerán enormemente al mejorarse la respuesta de la aplicación y reducirse el consumo de batería.

Puedes consultar y/o descargar el código completo de los ejemplos desarrollados en este artículo accediendo a la página del curso en GitHub.