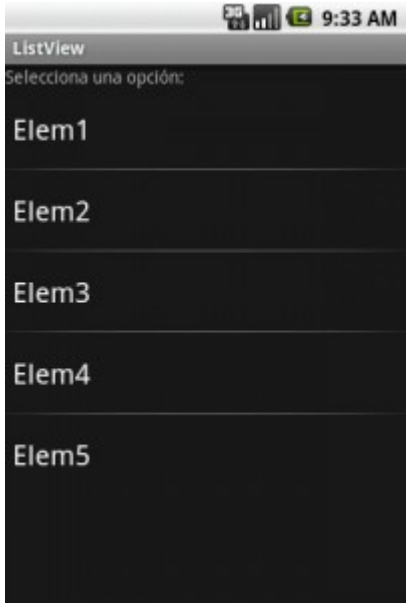
Controles de Selección

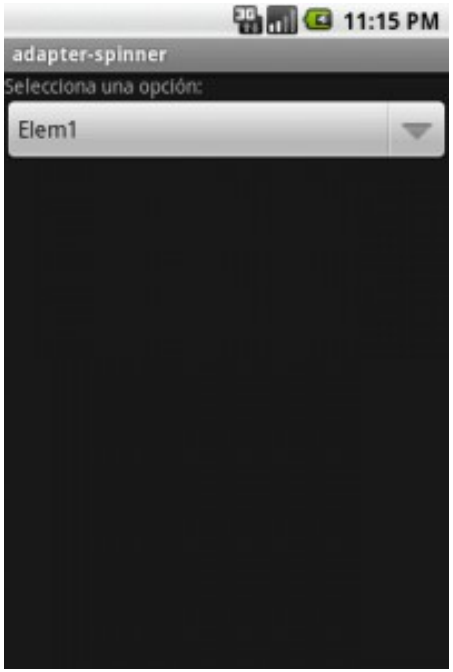
El modelo de datos asociado

CONTROLES DE SELECCIÓN

Definidos directamente en ficheros XML




```
<ListView
android:id="@+id/lista1"
android:prompt="@string/lista1_prompt"
android:entries="@array/lista1_entries"
android:layout_width="match_parent"
android:layout_height="wrap_content"/>
```




```
<Spinner android:id="@+id/spinner1"
android:prompt="@string/spinner1_prompt"
android:entries="@array/spinner1_entries"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
```

CONTROLES DE SELECCIÓN

Definidos directamente en ficheros XML

android:id

• Permite ser referenciado en el codigo Java y asignarle un manejador de evento.

android:prompt

•permite mostrar un texto sobre el spinner al hacer click sobre él.

android:entries

• un fichero XML con las entradas donde se define las opciones de elección.

Puede esta en el fichero *strings.xml* o definir un fichero separado (*arrays.xml*)

```
<string-array name="some_name">
<item>choice 1</item>
<item>choice 2</item>
…
</string-array>
```

```
<ListView
android:id="@+id/lista1"
android:prompt="@string/lista1_prompt"
android:entries="@array/lista1_entries"
android:layout_width="match_parent"
android:layout_height="wrap_content"/>

<Spinner android:id="@+id/spinner1"
android:prompt="@string/spinner1_prompt"
android:entries="@array/spinner1_entries"
android:layout_width="fill_parent"
```

XML: Fichero de opciones (Parte de res/layout/activity_lista.xml)

res/layout/activity_lista.xml

```
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/lista1_prompt"/>

<ListView
android:id="@+id/lista1"
android:prompt="@string/lista1_prompt"
android:entries="@array/lista1_entries"
android:layout_width="match_parent"
android:layout_height="wrap_content"/>
```

```
<string name="list1_prompt">
Android Vendors (Choices from XML)
</string>

<string-array name="lista1_entries">
<item>Acer</item>
<item>Dell</item>
<item>HTC</item>
<item>Huawei</item>
<item>Kyocera</item>
<item>LG</item>
<item>Motorola</item>
<item>Nexus</item>
<item>Samsung</item>
<item>Sony Ericsson</item>
<item>T-Mobile</item>
<item>Neptune</item>
</string-array>

<string name="list_message_template">
You selected \'%s\', 21 </string>
```

Lógica de aplicación en Java

```
public class ListaOpcionesXml extends Activity {
  @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_lista_opciones_xml);
        ListView lista1 = (ListView)findViewById(R.id.lista1);
         lista1.setOnItemClickListener(new ListaInfo());
   }
   private void showToast(String text) {
      Toast.makeText(this, text, Toast.LENGTH_LONG).show();
   }
```

```
      private class ListaInfo implements AdapterView.OnItemClickListener {
           @Override
          public void onItemClick(AdapterView<?> lista, View selectedView,
                                  int selectedIndex, long id) {
              String selection =  lista1.getItemAtPosition(selectedIndex).toString();
              showToast(selection);
           }
       }
       public void onNothingSelected(AdapterView<?> lista) {
           // Won't be invoked unless you programmatically remove entries
       }} de la clase interna
```

Controles de selección definidas en código Java

Modelo de datos asociado A LOS CONTROLES DE SELECCIÓN: Adaptadores en Android (*adapters*):

- Para los desarrolladores de java que hayan utilizado *Swing*, el concepto de *adaptador* les resultará familiar.

- Un adaptador representa algo así como una interfaz común al modelo de datos que existe por detrás de todos los controles de selección que hemos comentado.

- Dicho de otra forma, todos los controles de selección accederán a los datos que contienen a través de un adaptador.

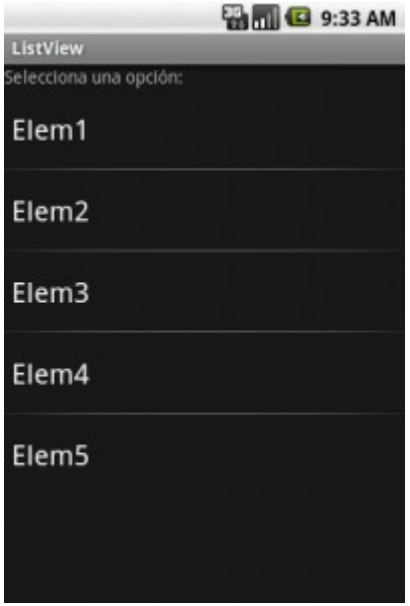
Controles de selección

Adaptadores en Android (*adapters*):

- **ArrayAdapter.** Es el más sencillo de todos los adaptadores, y provee de 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.
- **SimpleCursorAdapter.** Se utiliza para mapear las columnas de un cursor sobre los diferentes elementos visuales contenidos en el control de selección.

Control de selección (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.



Definición del control en XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android

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

Control de selección (ListView)

Gestion del control en la lógica de la aplicación:

```
public class PruebaLista extends Activity{
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.spinner);
      Spinner spinner1 = (Spinner) findViewById(R.id.spinner1);
      final TextView texto1 = (TextView) findViewById(R.id.texto1);

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);

ListView lstOpciones = (ListView)findViewById(R.id.LstOpciones);
 lstOpciones.setAdapter(adaptador);
```

Controles de selección

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

Tenemos 3 parámetros:

- El *contexto*, que normalmente será simplemente una referencia a la actividad donde se crea el adaptador.

- El ID del *layout* sobre el que se mostrarán los datos del control. En este caso le pasamos el ID de un layout predefinido en Android (android.R.layout. simple_list_item_1), formado únicamente por un control TextView, pero podríamos pasarle el ID de cualquier layout de nuestro proyecto con cualquier estructura y conjunto de controles, más adelante veremos cómo.

- El *array* que contiene los datos a mostrar.

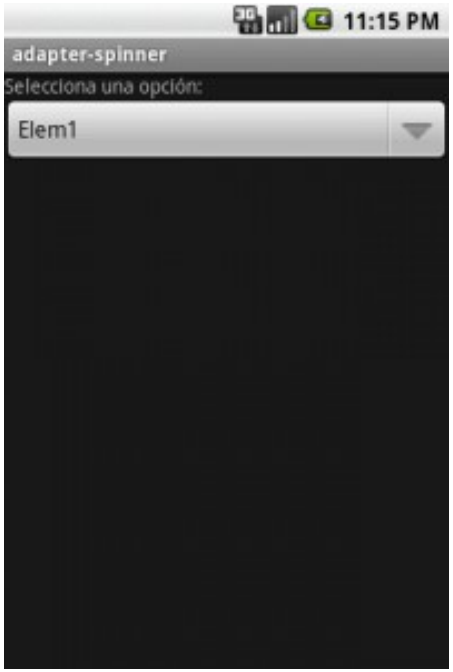
Controles de selección (II)

- 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:

```
lstOpciones.setOnItemClickListener(new
    OnItemClickListener() {

    public void onItemClick(AdapterView<?> a, View v, int
     position, long id) {
        //Acciones necesarias al hacer click
    }
});
```

Controles de selección




```
ArrayAdapter<String> adaptador =    new
ArrayAdapter<String>(this,   android.R.layout.simple_spinner_item, dato
```

```
adaptador.setDropDownViewResource(
      android.R.layout.simple_spinner_dropdown_item);
```

Control de selección (Spinner)

Propiedades del control en XML:

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

Para enlazar el adaptador a este control:

```
final Spinner cmbOpciones = (Spinner)findViewById(R.id.CmbOpciones);
 adaptador.setDropDownViewResource(
       android.R.layout.simple_spinner_dropdown_item);
 cmbOpciones.setAdapter(adaptador);
```

Controles de selección (Spinner)

- Uno de los parámetros que le pasábamos era el ID del layout que utilizaríamos para visualizar los elementos del control. Sin embargo, en el caso del control Spinner, este layout tan sólo se aplicará al elemento seleccionado en la lista, es decir, al que se muestra directamente sobre el propio control cuando no está desplegado.
- El funcionamiento normal del control Spinner incluye entre otras cosas mostrar una lista emergente con todas las opciones disponibles. Pues bien, para personalizar también el aspecto de cada elemento en dicha lista emergente tenemos el método setDropDownViewResource(*ID_layout*), al que podemos pasar otro ID de layout distinto al primero sobre el que se mostrarán los elementos de la lista emergente. En este caso hemos utilizado otro layout predefinido an Android para las listas desplegables (android.R.layout.simple_spinner_dropdown_item).

Controles de selección (Spinner)

- En cuanto a los eventos lanzados por el control Spinner, el más utilizado será el generado al seleccionarse una opción de la lista desplegable, **onItemSelected.** Para capturar este evento se le asigna su controlador mediante el método **setOnItemSelectedListener():**

```
cmbOpciones.setOnItemSelectedListener(
      new AdapterView.OnItemSelectedListener() {
      public void onItemSelected(AdapterView<?> parent,
         android.view.View v, int position, long id) {
            lblMensaje.setText("Seleccionado: " + datos[position]);
      }
       public void onNothingSelected(AdapterView<?> parent) {
         lblMensaje.setText("");
      }
});
```

- Para este evento se definen dos métodos
  - **onItemSelected** que será llamado cada vez que se selecciones una opción en la lista.
  - **onNothingSelected**)que se llamará cuando no haya ninguna opción seleccionada (esto puede ocurrir por ejemplo si el adaptador no tiene datos).

Controles de selección: Ejemplo

```
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;
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
 <!-- este laout lo llamamos:listitem_titular.xml“: -->
<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="fill_parent"
      android:layout_height="wrap_content"
      android:textStyle="bold"
      android:textSize="20px" />

<TextView android:id="@+id/LblSubTitulo"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:textStyle="normal"
      android:textSize="12px" />
</LinearLayout>
```

Controles de selección: Ejemplo

```
class AdaptadorTitulares extends ArrayAdapter {

    Activity context;

    AdaptadorTitulares(Activity context) {
        super(context, R.layout.listitem_titular, datos);
        this.context = context;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = context.getLayoutInflater();
        View item = inflater.inflate(R.layout.listitem_titular, null);

        TextView lblTitulo = (TextView)item.findViewById(R.id.LblTitulo);
        lblTitulo.setText(datos[position].getTitulo());

        TextView lblSubtitulo = (TextView)item.findViewById(R.id.LblSubTitulo);
        lblSubtitulo.setText(datos[position].getSubtitulo());

        return(item);
    }
}
```

Controles de selección : Ejemplo

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()`).

Una vez tenemos definido el comportamiento de nuestro adaptador la forma de proceder en la actividad principal será análoga a lo ya comentado,

```
private Titular[] datos =
    new Titular[]{
     new Titular("Título 1", "Subtítulo largo 1"),
     new Titular("Título 2", "Subtítulo largo 2"),

//...
//...

AdaptadorTitulares adaptador =
        new AdaptadorTitulares(this);

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

lstOpciones.setAdapter(adaptador);
```

Controles de selección : Ejemplo

```
private Titular[] datos =
   new Titular[]{
      new Titular("Título 1", "Subtítulo largo 1"),
      new Titular("Título 2", "Subtítulo largo 2"),
      new Titular("Título 3", "Subtítulo largo 3"),
      new Titular("Título 4", "Subtítulo largo 4"),
      new Titular("Título 5", "Subtítulo largo 5")};

//...
//...

AdaptadorTitulares adaptador =
       new AdaptadorTitulares(this);

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

lstOpciones.setAdapter(adaptador);
```

Controles de selección: Mejoras

- En el método getView() la forma normal de proceder consiste en “inflar” nuestro layout XML personalizado para crear todos los objetos correspondientes y posteriormente acceder a dichos objetos para modificar sus propiedades.
- Hay que tener en cuenta que esto se hace todas y cada una de las veces que se necesita mostrar un elemento de la lista en pantalla, se haya mostrado ya o no con anterioridad, ya que Android no “guarda” los elementos de la lista que desaparecen de pantalla (por ejemplo al hacer scroll sobre la lista).
- El efecto de esto es obvio, esto puede suponer la creación y destrucción de cantidades ingentes de objetos, es decir, que la acción de inflar un layout XML puede ser bastante costosa.

Controles de selección: Mejoras

- Android nos propone un método que permite reutilizar algún layout que ya hayamos inflado con anterioridad.
- ¿Pero cómo podemos reutilizar estos layouts “obsoletos”? Siempre que exista algún layout que pueda ser reutilizado éste se va a recibir a través del parámetro convertView del método getView().

```
public View getView(int position, View convertView, ViewGroup parent)
{  View item = convertView;
    if(item == null)
   {  LayoutInflater inflater = context.getLayoutInflater();
      item = inflater.inflate(R.layout.listitem_titular, null);
   }
……}
```

Controles de selección: Mejoras

- Hay otras dos instrucciones relativamente costosas que se siguen ejecutando en todas las llamadas. Me refiero a la obtención de la referencia a cada uno de los objetos a modificar mediante el método findViewById().
- Para mejorar rendimiento podemos aprovechar que estamos "guardando" un layout anterior para guardar también la referencia a los controles que lo forman.
- Nos definimos una clase *ViewHolder* con aquellos atributos con referencia a cada uno de los controles que tengamos que manipular :

```
static class ViewHolder {
   TextView titulo;
   TextView subtitulo;
}
```

Controles de selección: Mejoras

```
public View getView(int position, View convertView, ViewGroup parent)
{    View item = convertView;
   ViewHolder holder;
    if(item == null)
   {   LayoutInflater inflater = context.getLayoutInflater();
      item = inflater.inflate(R.layout.listitem_titular, null);
       holder = new ViewHolder();
      holder.titulo = (TextView)item.findViewById(R.id.LblTitulo);
      holder.subtitulo = (TextView)item.findViewById(R.id.LblSubTitulo);
       item.setTag(holder);
   }
   else
      {       holder = (ViewHolder)item.getTag();    }
   holder.titulo.setText(datos[position].getTitulo());
   holder.subtitulo.setText(datos[position].getSubtitulo());
    return(item);
```