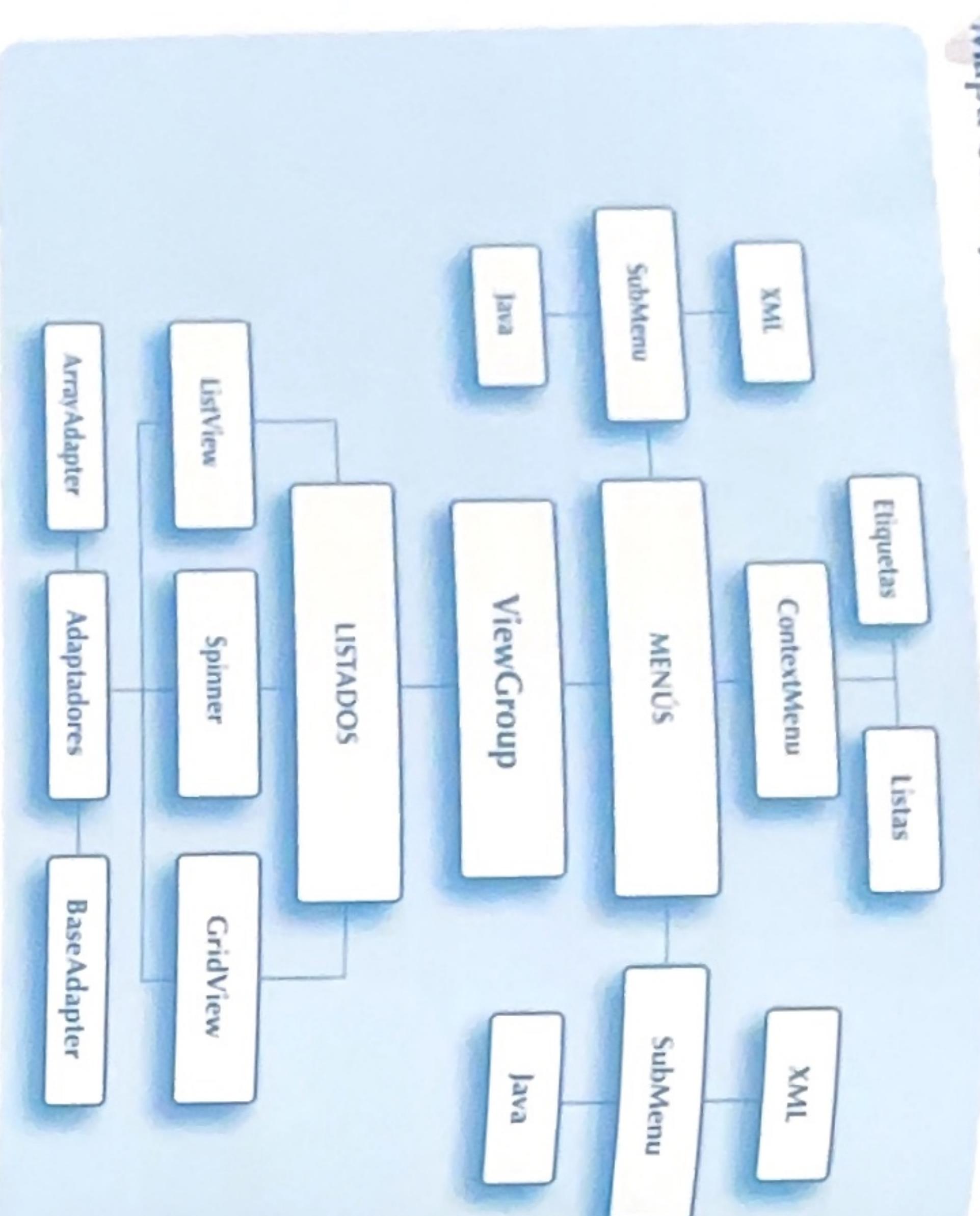


Mapa conceptual



6.1. Introducción

Tras el trabajo con controles básicos y después de haber estudiado los layouts, en este capítulo se va a avanzar en objetos ViewGroup. Debemos recordar que estos objetos son utilizados para la creación de una jerarquía de objetos View. Pueden ser instanciados de igual manera que los vistos anteriormente, desde XML o en código Java.

Un número importante de ellos son utilizados para mostrar la información al usuario en forma de lista (con variaciones visuales); son un conjunto de elementos ordenados por los que es posible navegar utilizando una barra de Scroll. Al seleccionar o pulsar uno de los elementos de la lista, se desencadenará un evento, que es posible controlar y al que se puede responder. Para dar un aspecto personalizado a los listados en cualquiera de sus variantes se deberá hacer uso de los adaptadores, que permitirán aumentar el número de objetos que se van a incluir en cada uno de los elementos que contiene el listado, así como realizar un diseño estéticamente a medida del que se desea que tengan dichos elementos.

Por otro lado, como cualquier entorno que disponga de interfaz gráfica, los menús son de extraordinaria utilidad en los procesos de selección entre opciones. Android tiene variantes de este tipo de objetos, algunas ya desaprobadas y otras que han evolucionado especialmente con la llegada de Material Design. Aquí se verán algunos de los más importantes.

6.2. Los listados

En este punto del capítulo, se aprenderá a crear listas en Android y a manejar sus diferentes elementos, trabajando con tres de los objetos ViewGroup más frecuentes. Además, se iniciará el estudio en el interesante mundo de los adaptadores. En la figura 6.1 se muestra cuál sería el aspecto que tendría cada uno de estos objetos si se sigue el código de cada apartado.



6.3. ListView

Este objeto visualiza una lista deslizable verticalmente de varios elementos; cada uno de ellos puede ser seleccionado sobre el propio control. En caso de disponer de más elementos de los que es posible mostrar en pantalla, aparecerá una barra de Scroll que permitirá acceder a los elementos no visibles. El código XML del objeto sobre el que construir el listado es el siguiente.

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

Menú contextual. Ventana con un grupo de opciones que se abre cuando hacemos click sobre un objeto, generalmente con el botón secundario del ratón o con una pulsación sostenida en un dispositivo móvil.

POJO (Plain Old Java Object). Instancia de una clase que ni extiende ni implementa nada

Podemos utilizar objetos de este tipo relacionándolos con el adaptador para enlazar el componente del interfaz con la colección de datos.

ScrapViews. Vista secundaria que ha salido de pantalla, se mantiene en memoria y puede volverse a usar, posteriormente, tan solo actualizándose (reciclando).

Investiga

Podemos utilizar también los siguientes atributos para modificar la estética del ListView. Práctica con cada uno de ellos.

- *android:divider*: permite configurar el color de la línea divisoria.
- *android:dividerHeight*: configura el grosor de la línea divisoria.
- *android:footerDividersEnabled*: habilita el divisor que va delante del "footer" (último).
- *android:headerDividersEnabled*: habilita el divisor que va detrás del "header" (primero).

Los pasos que se van a seguir para poder manejar la lista desde el código Java son: instanciar el objeto, declarar un array de valores (string), crear un adaptador que nos permita rellenar la lista y asociar el adaptador a la lista.

```
ListView Listado = (ListView) findViewById(R.id.listView);
final String[] datos = new String[]{"Elemento1", "Elemento2", "Elemento3", "Elemento4", "Elemento5"};
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, datos);
listado.setAdapter(adaptador);
```

TOMA NOTA

Un adaptador suele ser un objeto de la subclase BaseAdapter, que posibilita el convertir los datos en diferentes elementos de la lista. Android proporciona algunos adaptadores estándar, como ArrayAdapter y CursorAdapter. El primero permite manejar datos cuyo origen es un array, mientras que en el segundo caso los datos provendrán de una base de datos.

Para obtener la información del elemento pulsado se utiliza el manejador de eventos *setOnItemClickListener*, que podrá controlar el evento *onItemClick*, al que se le pasará como parámetro el adaptador y devolverá la posición del elemento tocado.

```
listado.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int posicion, long id) {
        //comenzar al texto del elemento pulsado
    }
});
```

Una vez se tenga esta posición, se puede obtener el contenido del elemento en sí mediante dos formas, una de ellas pidiendo al listado (parent) que devuelva el elemento que se encuentra en la posición tocada, como se muestra en este ejemplo:

```
String elemento = (String) parent.getItemAtPosition(posicion);
```

O bien obteniendo el adaptador del listado y pidiéndoselo a este:

```
String elemento = (String) parent.getAdapter().getItem(posicion);
```

RECUERDA

- ✓ Realiza una aplicación cuya actividad (Activity) tenga objetos dentro de un LinearLayout vertical, el primero un ListView y el segundo un TextView. Rellena el ListView con el nombre de diez países europeos. Cuando se ejecute el TextView, debe tomar el nombre del elemento tocado del listado.

Actividad propuesta 6.1

Muestra los datos a modo de rejilla bidimensional e incluye automáticamente un Scroll para cuando los datos ocupen más tamaño que las posibilidades de la pantalla. Al igual que en el ListView, los datos provienen de unListAdapter o incluso de un Adapter personalizado. Atributos importantes para este objeto son:

- *android:numColumns*: número de columnas que deseamos establecer en el GridView.
- *android:columnWidth*: define el ancho de cada columna de la cuadrícula.
- *android:verticalSpacing*: separación vertical entre las filas del GridView.
- *android:horizontalSpacing*: separación horizontal entre las columnas del GridView.

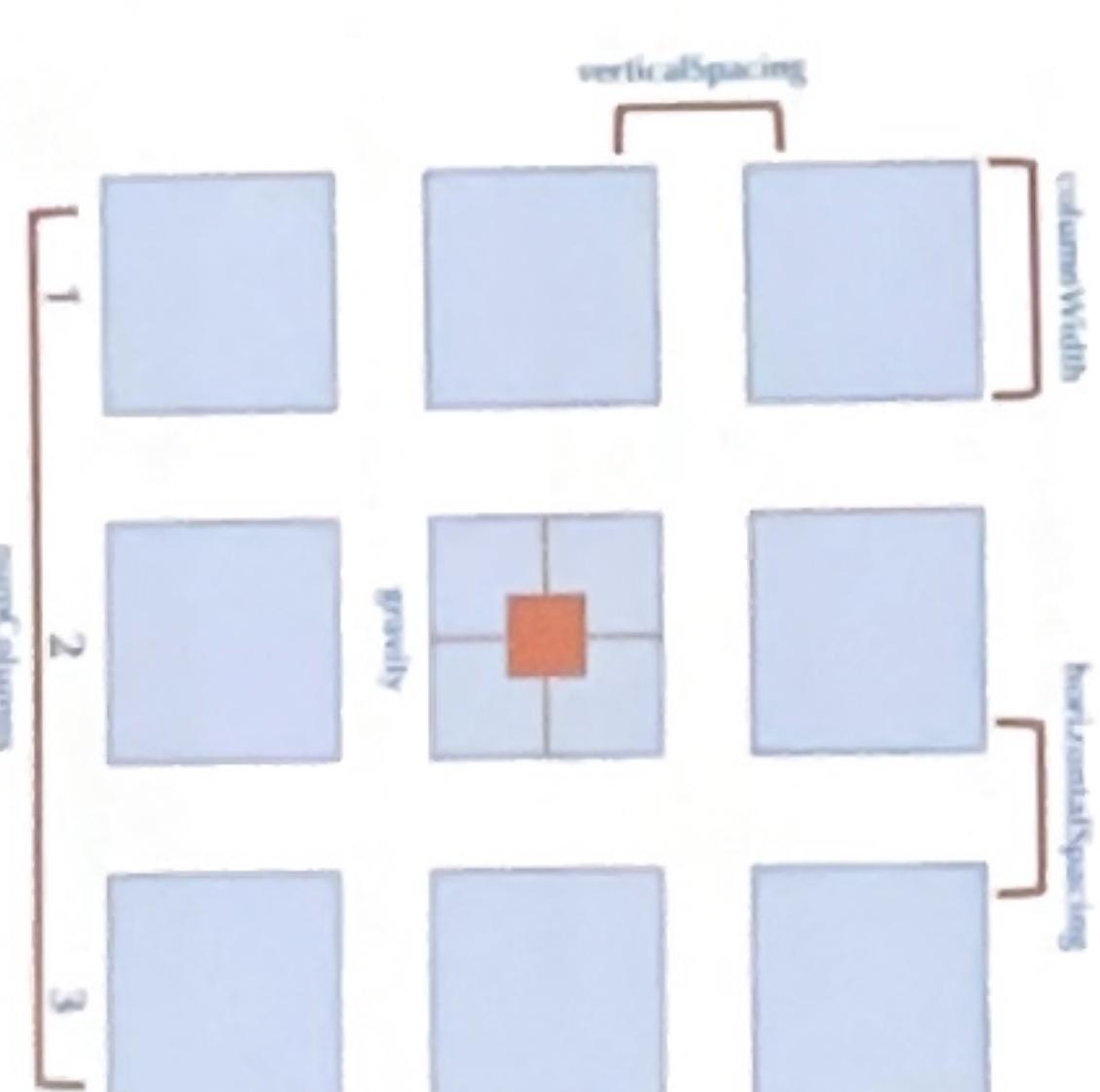


Figura 6.2 Esquema de organización de GridView.

- android:stretchMode* define el modo en que se extenderán las columnas.
- android:gravity*: define el posicionamiento del contenido en cada celda.

android:stretchMode define el modo en que se extenderán las columnas, *android:gravity*: define el posicionamiento del contenido en cada celda.



Investiga

Las posibilidades de *stretchMode* son:

- *None*: ninguna extensión.
- *spacingWidth*: se extiende al espacio entre cada columna.
- *columnWidth*: se hace un reparto equitativo del espacio.
- *spacingWidthUniform*: se hace un reparto uniforme.
- *spacingWidthFit*: que nos ajustará el número de columnas dependiendo de las dimensiones de la pantalla de nuestro dispositivo.

Otra posibilidad es usar el flag "*auto_fit*" que nos ajustará el número de columnas dependiendo de las dimensiones de la pantalla de nuestro dispositivo.

Investiga y practica con cada uno de ellos.

6.5. Spinner

Este objeto es una evolución del estudiado entre los objetos básicos del interface, tan solo que su aspecto puede ser mejorado mediante adaptadores personalizados. Similar en la construcción al anterior, este objeto muestra los datos en un listado desplegable de elementos, de los cuales el usuario puede seleccionar uno, que se mostraría al frente del Spinner.

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

Como en otras ocasiones, para manejar el Spinner desde Java se debe instanciar el objeto, declarar el array de valores (string) y asociarle un adaptador a la lista.

```
Spinner listado = (Spinner) findViewById(R.id.miSpinner);
final String[] datos = new String[]{"Elemento1", "Elemento2", "Elemento3", "Elemento4", "Elemento5"};
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, datos);
listado.setAdapter(adaptador);
```

Con un escuchador del tipo *setOnItemSelectedListener* se detectará el elemento pulsado.

```
listado.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int position, long id)
    {
        //Obtener el texto del elemento pulsado
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent)
    {
        //Obtener el texto del elemento pulsado
    }
});
```

Y para detectar el elemento tocado sería exactamente igual que en el ListView.

```
listado.setOnItemClickListener(new AdapterView.OnItemClickListener() {
```

```
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int posicion, long id) {
```

```
        //Obtener el texto del elemento pulsado
    }
});
```

Actividad propuesta 6.2

Modifica el código de la actividad realizada para ListView cambiando este objeto por un GridView de dos columnas, que tenga igual respuesta al tocar uno de los elementos.



Actividad propuesta 6.3

Realiza una aplicación similar a las dos anteriores, pero con Spinner.

6.6. Trabajando con adaptadores

Si bien la utilización del `simple_list_item_1` como adaptador es bastante sencilla, si se necesita incluir más de un elemento en el listado o se quisiera mejorar el aspecto visual del mismo (personalizando colores, fuentes o mostrando imágenes, enlaces web...), se tendría que elaborar un adaptador personalizado. En este apartado se va a aprender cómo funcionan los adaptadores y adaptarlos en algunas de sus variantes.

El entender la función de los adaptadores hará más fácil su programación e implementación en las aplicaciones. Para ello, se va a ver cómo funciona el reciclaje de las vistas.

Cuando se conecta un ListView a un adaptador, este se ocupa de crear las instantáneas de las filas necesarias hasta que ListView tenga suficientes elementos como para completar la altura que ocupa el listado en la pantalla, no siendo necesario preparar elementos adicionales en memoria.



Figura 6.3
Función del adaptador.

Si se interacciona con el listado y se desplazan los elementos hacia arriba o hacia abajo, aquellos elementos que salen de la pantalla quedarán en memoria para un uso posterior, incorporando tantas nuevas filas como las que han sido guardadas en memoria.

Las vistas que salen, denominadas `Views`, pueden volverse a utilizar posteriormente sin tener que inflarse, solo deben actualizarse (reciclarse), por tanto, el objeto de lista solo necesita mantener suficientes vistas en la memoria para completar el espacio asignado en el diseño, y algunas vistas reciclables adicionales.

Tras estos conceptos básicos sobre adaptadores, se va a crear uno que gestione un ListView (util también para GridView y Spinner), que mostrará dos líneas de texto con formato personalizado.

El primer paso será realizar una clase Java que gestione y suministre los datos. Esta deberá tener tantos atributos como se vayan a poner en el listado. Para este ejemplo podría servir la siguiente:

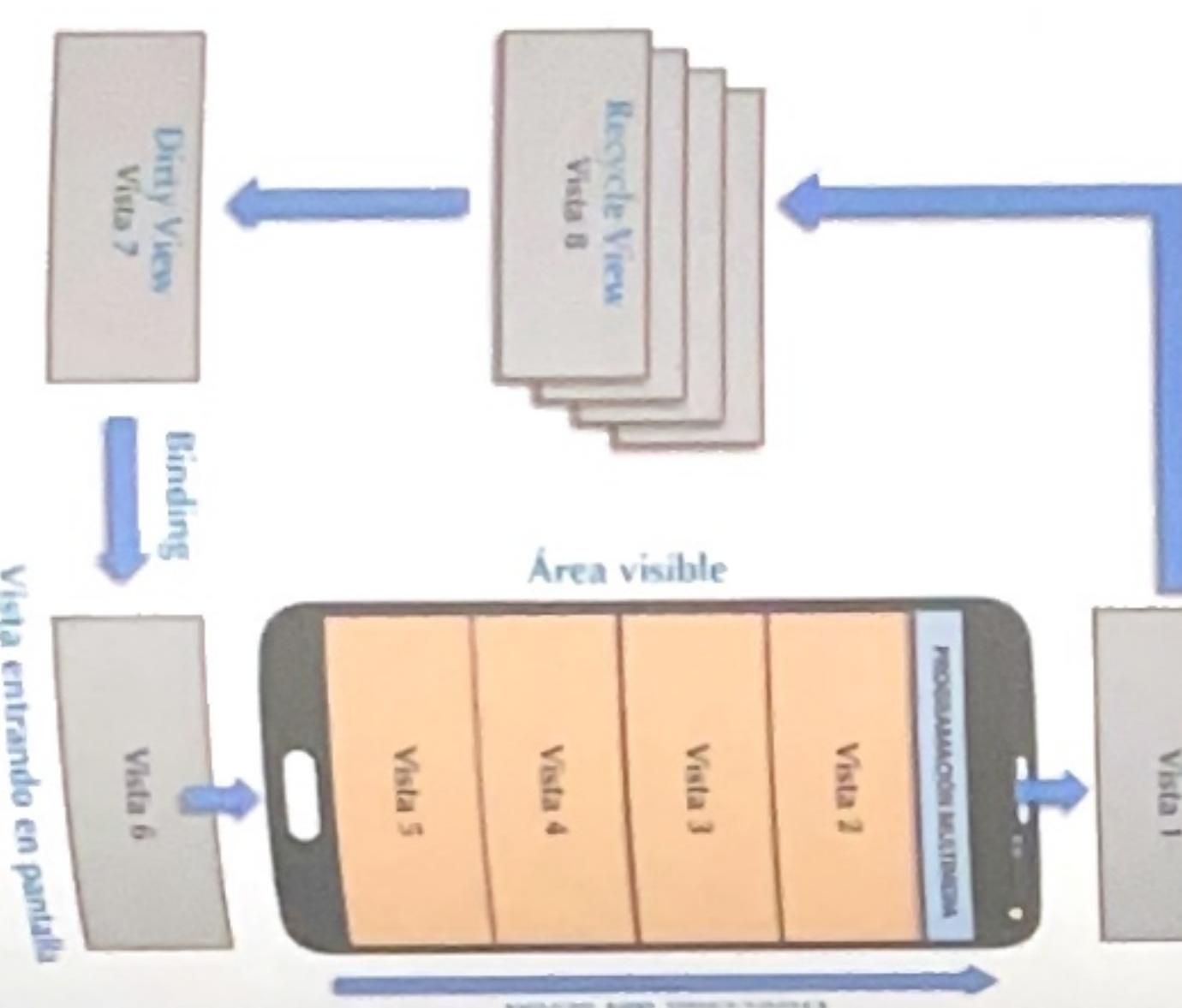


Figura 6.4
Dinámica de las vistas en pantalla.

A continuación, deberá crearse en la carpeta res/layout un fichero XML con el diseño que se quiere que tenga la vista de datos (cada elemento del listado). Un ejemplo simple podría ser el siguiente (lo llamaremos `elemento.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/miTexto1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold" />
    <TextView android:id="@+id/miTexto2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textStyle="normal"
        android:textSize="12sp" />
</LinearLayout>
  
```

Por último se creará el adaptador, para lo cual pueden ser utilizadas dos variantes: Adapter y BaseAdapter.

6.6.1. ArrayAdapter

Este es, quizás, el adaptador más sencillo porque convierte un ArrayList de objetos en vistas, cargadas en un contenedor, que en este caso es un ListView. Este adaptador se ocupa de localizar los datos para la lista y convertirlos en un objeto para la vista.

Para ello es conveniente que se cree una clase independiente, que extenderá de ArrayAdapter, a cuyo constructor se le pasará el contexto (que suele ser la actividad principal) y la clase que se ha creado para gestionar el array de objetos con los datos que se van a mostrar.

Dentro del constructor se llamará a la superclase, pasándole como parámetros el contexto, el layout que da formato a cada elemento de la lista y los datos. Además, se pasará la referencia del objeto datos como parámetro para otros métodos (previamente lo tendremos que haber

```

public class Datos {
    private String text1;
    private String text2;

    public Datos (String text1, String text2){
        text1 = text1;
        text2 = text2;
    }

    public String getTexto1(){
        return text1;
    }

    public String getTexto2(){
        return text2;
    }
}
  
```

Figura 6.5
Ejemplo de adaptador con dos ejemplos.



declarado). El aspecto que tendría hasta ahora esta clase, a la que se ha llamado *Adaptador*, es el siguiente:

```
public class Adaptador extends ArrayAdapter<Datos> {
    private Datos[] datos;
    private Context context;
    public Adaptador(Context context, int layout_elemento, Datos[] datos) {
        super(context, R.layout.elemento, datos);
        this.datos = datos;
    }
}
```

A continuación, y dentro de esta clase, se debe sobrescribir el método (*getView*) que se encarga de generar y llenar con los datos cada uno de los elementos del listado mostrado en la interfaz gráfica de cada elemento de la lista.

Este método será llamado cada vez que haya que mostrar un nuevo elemento de la lista. Se realizará el siguiente proceso. Lo primero que hará es “inflar” el layout que da formato a cada elemento (en este ejemplo, se ha llamado *elemento.XML*). Para esto, se instancia un objeto *LayoutInflater* (mostrado), se localiza su contexto (*getApplicationContext*) y se asocia la vista al elemento a “inflar”. Por último, queda tan solo referenciar los objetos que aparecen en el XML (*textol* y *text02*), que se llenarán con los datos que se encuentran en la posición que se va a mostrar del array.

```
public View getView(int position, View convertView, ViewGroup parent) {
    LayoutInflater mostrado = LayoutInflater.from(getApplicationContext());
    View elemento = mostrado.inflate(R.layout.elemento, parent, false);
    TextView textol = (TextView) elemento.findViewById(R.id.miTextol);
    textol.setText(datos[position].getTextol());
    TextView texto2 = (TextView) elemento.findViewById(R.id.miTexto2);
    texto2.setText(datos[position].getTexto2());
    return elemento;
}
```

Ya solo queda preparar la llamada desde el Java principal. Primero se prepara el array con la información para mostrar (en el ejemplo se mostraban dos cadenas de texto). Esto se puede hacer de varias maneras, una de ellas podría ser la siguiente:

```
Datos[] datos = new Datos[]{
    new Datos("Línea Superior 1", "Línea Inferior 1"),
    new Datos("Línea Superior 2", "Línea Inferior 2"),
    new Datos("Línea Superior 3", "Línea Inferior 3"),
    new Datos("Línea Superior 4", "Línea Inferior 4")};
```

Como es fácil imaginar, no es operativo incluir en la clase principal fuentes de datos con mucha información, sino que se debe recurrir a otras alternativas, como pueden ser una base de datos, fichero en memoria, base de datos.

A continuación, se incorpora el objeto *ListView* en el XML principal de la aplicación (que como se vio en el apartado anterior), se instancia y, por último, se crea el adaptador (que se creó una instancia del adaptador personalizado) y se asocia al *ListView*.

```
ListView listado = (ListView) findViewById(R.id.listView);
Adaptador miAdaptador = new Adaptador(this, datos);
listado.setAdapter(miAdaptador);
```

Ponerle una cabecera al listado es fácil, para ello se debe crear un XML con el aspecto que se quiere que tenga la cabecera. Como ejemplo, creamos un fichero llamado *cabecera.XML* (en la carpeta layout) con una sola línea y fondo amarillo.

```
<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="25dp"
        android:text="LISTADO EJEMPLO"
        android:textStyle="bold"
        android:background="#ffff00"
        android:gravity="center" />
</LinearLayout>
```

Ahora se insertará la cabecera (antes de aplicar el adaptador al listado), dos líneas de código, una que crea la vista de la cabecera y gestiona su inflado, y otra que añade la cabecera a la lista.

```
View miCabecera = getLayoutInflater().inflate(R.layout.cabecera, null);
listado.addHeaderView(miCabecera);
```

Para detectar el elemento pulsado, se utiliza un proceso similar al anteriormente visto. Se crea un escuchador al listado (*setOnItemClickListener*) para ese evento (*onItemClick*), que debe recibir cuatro parámetros, el control que contiene la lista (*AdapterView*), la vista del objeto pulsado (*View*), la posición del elemento pulsado (*position*) y el id del elemento pulsado.

Queda acceder a la vista asociada al adaptador y obtener el elemento situado en *position* usando *getItemAtPosition()* del que se puede extraer la información que interese.

```
listado.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> adaptador, View v, int position, long id) {
        String seleccionado = ((Datos) adaptador.getItemAtPosition(position)).getTextol();
    }
});
```

Investiga



Incorpora todos los objetos que deseas (direcciones web, CheckBox, RadioButton, imágenes...) a cada elemento de la lista. Para ello debes realizar un diseño del XML que gestiona su vista con el formato adecuado a tus necesidades y luego incorporar estos objetos al Java que gestiona los datos.

Lógicamente, la fuente de datos (en nuestro caso, el array) debe tener todos los elementos para mostrar. En caso de tratarse de imágenes, en el array debe aparecer la URL o recurso de memoria donde se encuentran.

6.6.2. BaseAdapter

Ahora se verá el uso de `BaseAdapter`, una clase muy utilizada para crear adaptadores en `ArrayList`. La construcción es muy similar al de `ArrayAdapter`, ya que previamente se ha de diseñar el proyecto que se quiere que tenga cada elemento de la vista de datos, así como una clase Java que gestione los datos (pueden servir los utilizados en `ArrayAdapter`).

Este adaptador extenderá de `BaseAdapter`, que tiene una serie de métodos que es importante conocer. Son estos:

- `int getCount()`
- `Object getItem(int position)`
- `long getItemId(int position)`
- `View getView(int position, View convertView, ViewGroup parent)`

• `View getView(int position, View convertView, ViewGroup parent)`

De ellos, `getCount()` devuelve el número total de elementos que se mostrarán en la lista. Para lo que tiene en cuenta el tamaño del array. Cuando un elemento de la lista esté preparado para mostrarse, es llamado el método `getView (int i, View view, ViewGroup convertView)` para mostrarlo, colaborando con la clase `LayoutInflater` para su mostrado en pantalla.

Para conocer el elemento que se encuentra en una posición específica dentro de la colección de datos, será necesario recurrir a `getItem (int i)`, mientras que `getItemId (position)` devuelve el id de posición del elemento.

Ahora el constructor del adaptador recibirá un `ArrayList` generado a partir del Java que contiene el suministro de datos (POJO). Y, al igual que con `ArrayAdapter`, será el método `getCount()` quien gestione el inflado del `ListView` con los datos aportados, obteniendo en cada caso los datos del array según la posición en la que se encuentre. El código final del adaptador quedaría de la siguiente forma:

```
public class Adaptador extends BaseAdapter {
    private ArrayList<Datos> datos;
    private Context contexto;
    public Adaptador (Context contexto, ArrayList<Datos> datos) {
        super();
        this.contexto = contexto;
        this.datos = datos;
    }
    @Override
    public View getView(int posicion, View view, ViewGroup parent) {
        LayoutInflator mostrado = LayoutInflater.from(contexto);
        View elemento = mostrado.inflate(R.layout.elemento, parent, false);
        TextView texto1 = (TextView) elemento.findViewById(R.id.miTexto1);
        texto1.setText(datos.get(posicion).getTexto1());
        TextView texto2 = (TextView) elemento.findViewById(R.id.miTexto2);
        texto2.setText(datos.get(posicion).getTexto2());
        return elemento;
    }
    @Override
    public int getCount() { return datos.size(); }
    @Override
    public Object getItem(int posicion) {return datos.get(posicion); }
    public long getItemId(int posicion) { return posicion; }
}
```

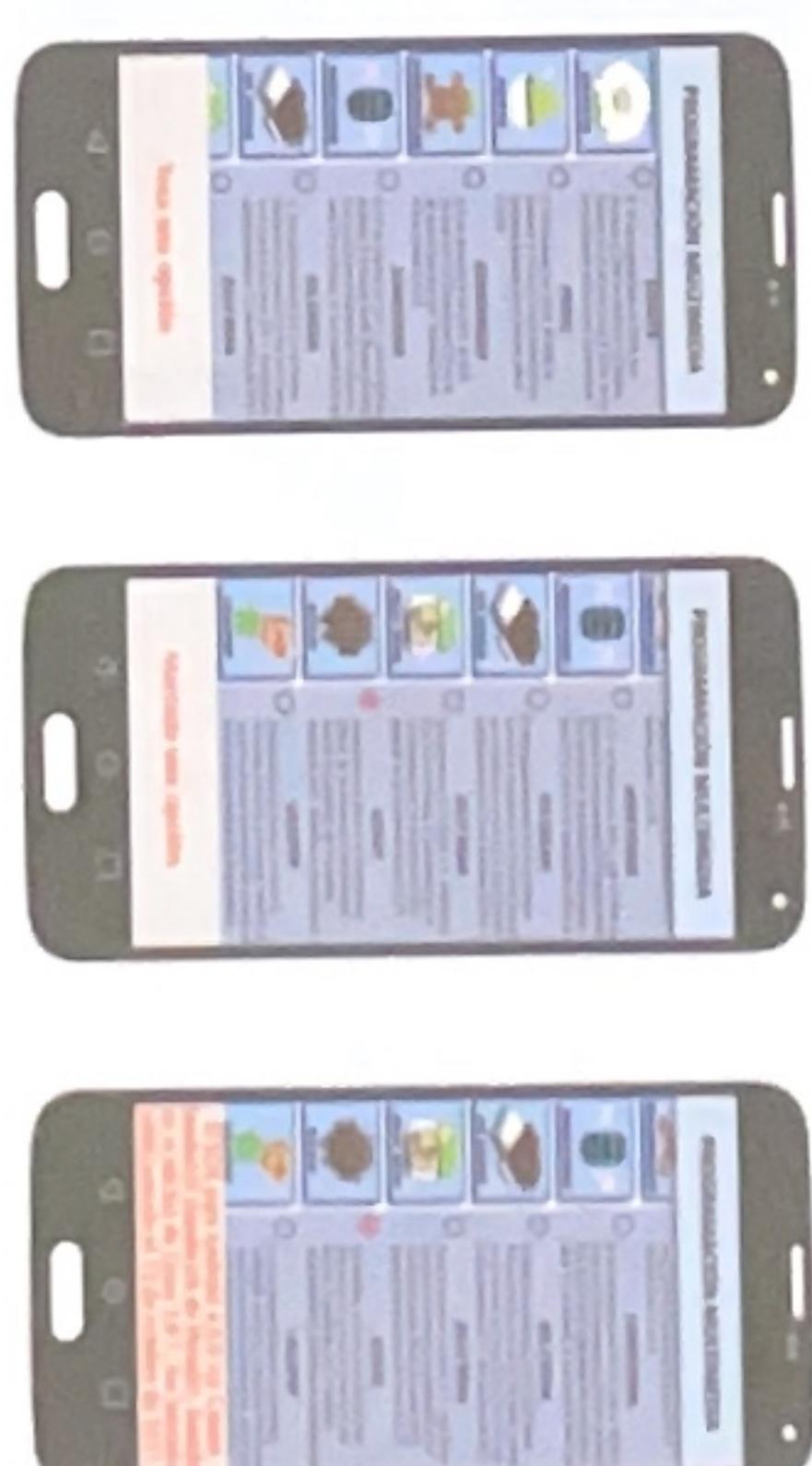
Además, se han de hacer modificaciones en el Java principal, ya que ahora se suministran los datos en un `ArrayList`, quedando esa parte del código del siguiente modo:

```
ArrayList<Datos> datos = new ArrayList<Datos>();
datos.add(new Datos("Línea Superior 1", "Línea Inferior 1"));
datos.add(new Datos("Línea Superior 2", "Línea Inferior 2"));
datos.add(new Datos("Línea Superior 3", "Línea Inferior 3"));
datos.add(new Datos("Línea Superior 4", "Línea Inferior 4"));
```

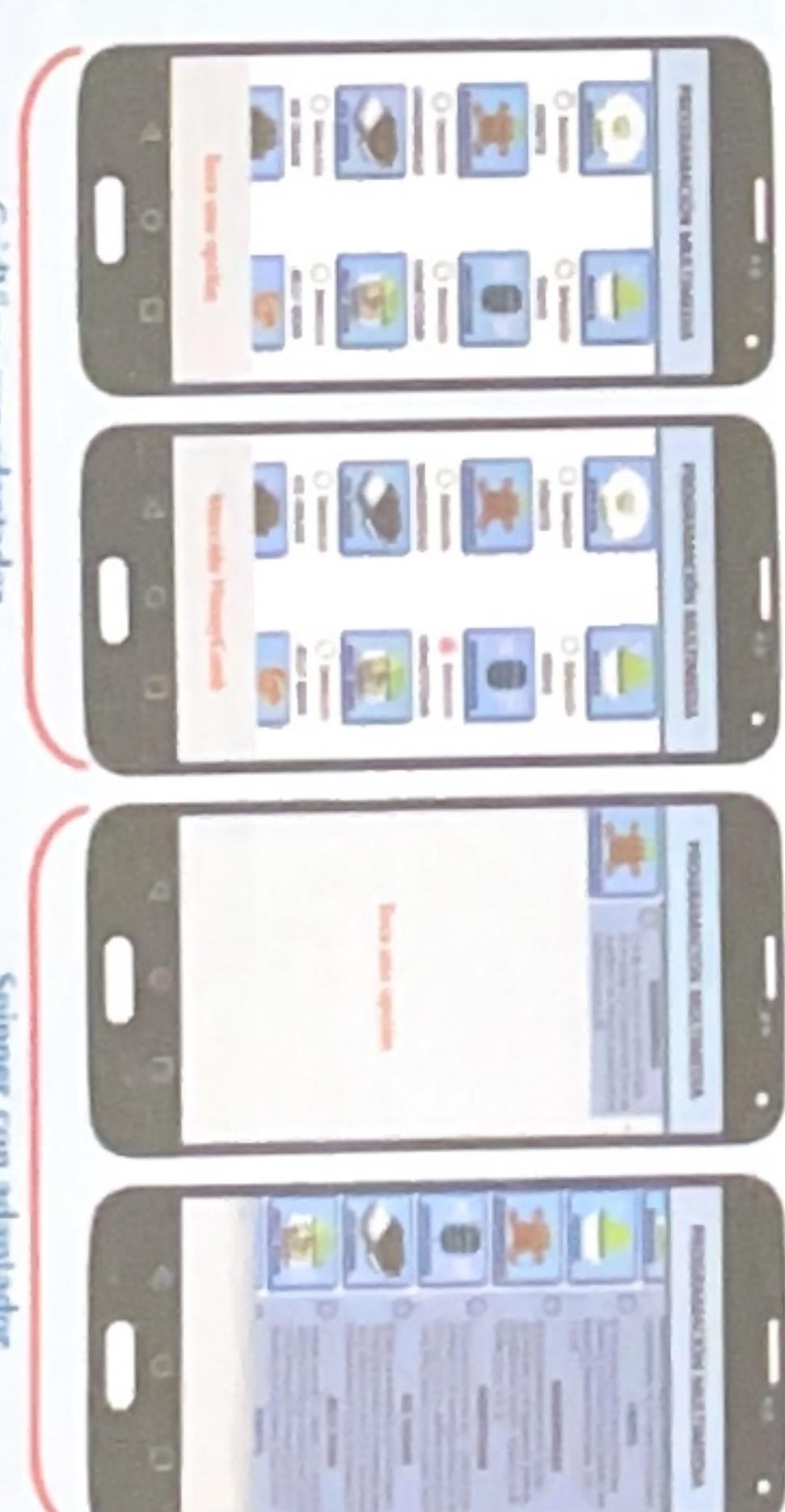
Actividad propuesta 6.4

En la plataforma de Editorial Síntesis encontrarás un tutorial guiado para realizar una actividad que implemente un adaptador (`BaseAdapter`) en un `ListView`, en el que cada elemento contendrá: imagen, título, contenido y botón de radio.

Recuerda que para construir este proyecto necesitarás, además de la `Activity` principal (que incluirá una clase POJO para los datos), otra clase con el adaptador y dos ficheros XML: uno será la vista de la clase principal (que tendrá el `ListView`) y otro dará forma a los elementos del listado. El tutorial te ayudará también a controlar los eventos de respuesta al `RadioButton` y al texto del `ListView`, tal y como aparecen en la siguiente imagen.



Terminada la actividad con `ListView` y adaptadores, intenta realizar las siguientes prácticas (`GridView` y `Spinner`), siguiendo un proceso muy similar al anterior.



GridView con adaptador

Spinner con adaptador

6.7. OptionsMenu

Se trata del más simple y clásico menú, que se despliega al pulsar una tecla u opción de pantalla. Puede construirse incorporando las opciones en un fichero XML (con el nombre `descarga_menu.xml`) ubicado en `res/menu`:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/mnOp1" android:title="Opción de menú A"></item>
    <item android:id="@+id/mnOp2" android:title="Opción de menú B"></item>
    <item android:id="@+id/mnOp3" android:title="Opción de menú C"></item>
</menu>
```

Para mostrarlo ("inflarlo") se ha de utilizar en Java el siguiente método:

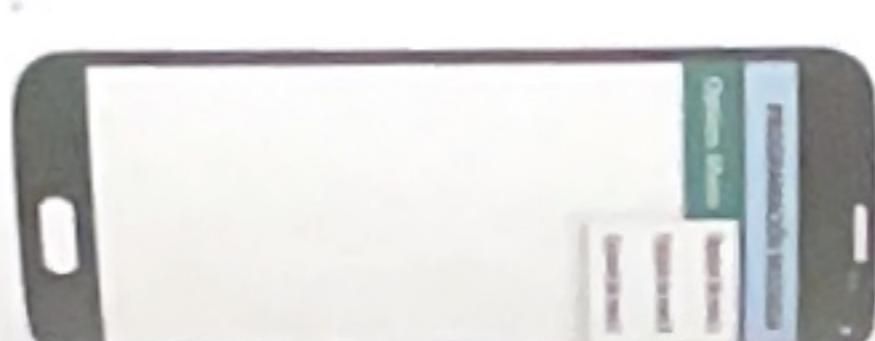
```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    public boolean onCreateOptionsMenu(Menu menu, menu);
    getMenuInflater().inflate(R.menu.opciones, menu);
    return true;
}
```

Y solo queda asociarle un escuchador que responda a alguno de los identificadores que han asignado a cada opción de menú.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    String mensaje = "";
    switch (item.getItemId()) {
        case R.id.MnOp1:
            // Código para realizar
            return true;
        case R.id.MnOp2:
            // Código para realizar
            return true;
        case R.id.MnOp3:
            // Código para realizar
            return true;
    }
}
```

Figura 6.6
Ejemplo de OptionsMenu.



6.8. Submenú

Para crear subopciones dentro de un menú, se puede proceder igual que en el caso anterior, definiéndolo desde XML, para lo cual debe crearse un nuevo menú dentro de las etiquetas del ítem al que se quieren asociar opciones secundarias, como en el ejemplo siguiente:

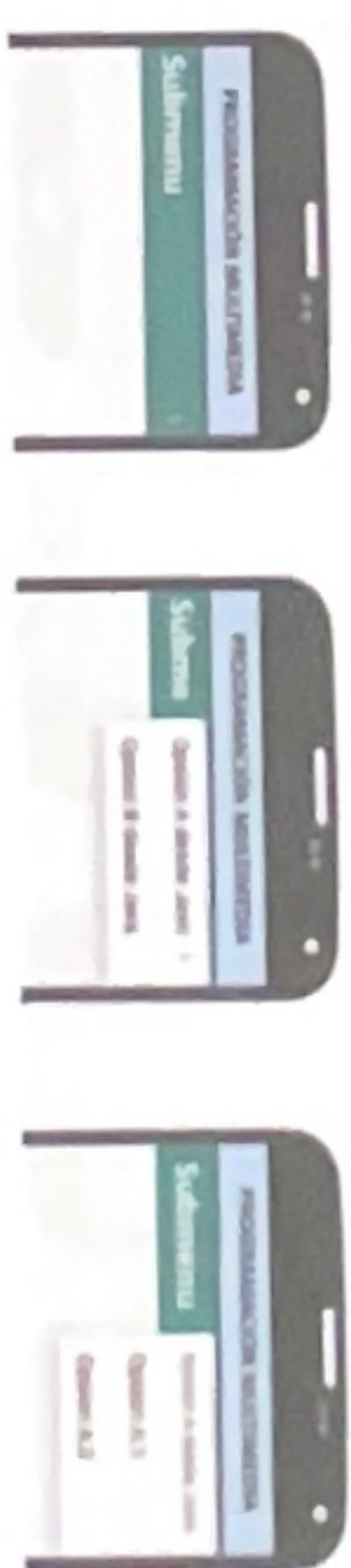
```
<item
    android:id="@+id/mnOp2"
    android:title="Opción de menú B">
<menu>
    <item android:id="@+id/mnOp2_1"
        android:title="Opción B.1" />
    <item android:id="@+id/mnOp2_2"
        android:title="Opción B.2" />
</menu>
```

`</item>`

Para hacerlo desde Java, se crea una instancia de la clase `SubMenu` y se añade a la opción de menú de la que dependa. Luego se incorpora cada una de las opciones de submenu.

```
public boolean onCreateOptionsMenu(Menu menu) {
    SubMenu smnu = menu.addSubMenu(Menu.NONE, MnOp1, Menu.NONE, "Opción A.1");
    smnu.add(Menu.NONE, MnOp1_1, Menu.NONE, "Opción A.1.1");
    smnu.add(Menu.NONE, MnOp1_2, Menu.NONE, "Opción A.1.2");
    menu.add(Menu.NONE, MnOp2, Menu.NONE, "Opción B desde Java");
    return true;
}
```

Figura 6.7
Submenús en Android.



Actividad propuesta 6.5

Realiza las siguientes actividades de menú (en XML) y submenu (en Java) con escuchadores a los eventos según aparecen en las imágenes.

Para crearlo en su totalidad desde Java, se han de introducir los identificadores y luego dirigir cada una de las opciones de menú con el método `OnCreateOptionsMenu`.

```
private static final int MnOp1 = 1;
private static final int MnOp2 = 2;
//...
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(Menu.NONE, MnOp1, Menu.NONE, "Opción A desde Java");
    menu.add(Menu.NONE, MnOp2, Menu.NONE, "Opción B desde Java");
}
}
```

En el escuchador, los casos del `Switch` serán 1 y 2, los valores que se le han dado a los `id` identificadores de las opciones.

Ten en cuenta

- ✓ Los escuchadores que responden a cada uno de los eventos de las opciones de submenú se construyen de igual forma que las de las opciones principales.

6.9. Menú contextual

Son opciones de menú que aparecen en la interfaz de usuario cuando se realiza una pulsación larga sobre algún elemento de la pantalla. Para construirlo se debe asociar el menú contextual a algún objeto de pantalla, frecuentemente a una etiqueta de texto.

```
TextView etiqueta = (TextView) findViewById(R.id.texto);
registerForContextMenu(etiqueta);
```

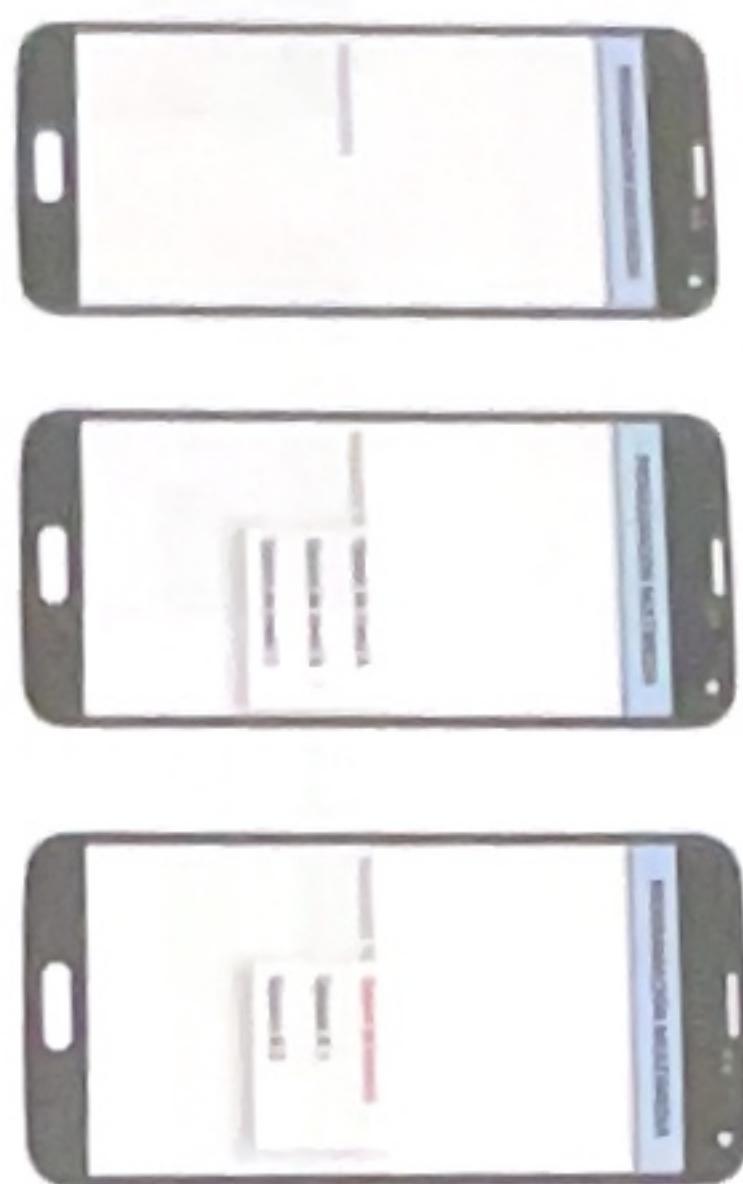


Figura 6.8
Menús
contextuales.

Es necesario, también, definir las opciones de menú y, en su caso, submenu, que puede hacerse de cualquiera de las formas anteriores (en este ejemplo, se ha utilizado el mismo fichero XML de las veces anteriores). Luego se trabajará con una clase específica de este menú, onCreateOptionsMenu, en el que se debe construir e inflar el menú contextual.

Ahora se han de crear tantos ficheros XML de menú como menús contextuales distintos se desea que aparezcan. En este caso se han definido tres (los puedes descargar de la plataforma). A manera de ejemplo, aquí se muestra uno de ellos.

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/ListaOpA1"
        android:title="OPCIÓN 1 DE MENÚ DE LISTA A"></item>
    <item android:id="@+id/ListaOpA2"
        android:title="OPCIÓN 2 DE MENÚ DE LISTA A"></item>
</menu>
```

- ✓ Los identificadores de las opciones siempre deben ser distintos.

RECUERDA

En el Java se define el array con los datos que se mostrarán, se instancia la lista, se crea el adaptador, se le asocia a la lista y se registra la lista para que tenga un menú contextual.

```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuItemInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    getMenuInflater().inflate(R.menu.menu, menu);
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
{
    inflater.inflate(R.menu.menu, menu);
}
```

El escuchador para este menú se construye de manera similar a los anteriores, pero se sustituye onOptionsItemSelected por onContextItemSelected.

Recurso web

En este código QR encontrarás un vídeo de Codeloverlatino (Jorge Luis Vilavicencio Correa), en el que seguir un tutorial para la creación de menús contextuales en Android.



www

```

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuItemInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    getMenuInflater().inflate(R.menu.menu_list1, menu);
    registerForContextMenu(menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
    AdapterView.AdapterContextMenuInfo info2 = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
    int position = info2.info.position;
    switch (info.position) {
        case 0:
            LayoutInflater inflater = (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            View view = inflater.inflate(R.layout.item_menu, null);
            TextView textView = (TextView) view.findViewById(R.id.textView);
            textView.setText("Elemento " + position);
            textView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Toast.makeText(getApplicationContext(), "Elemento " + position + " seleccionado", Toast.LENGTH_SHORT).show();
                }
            });
            return true;
        case 1:
            LayoutInflater inflater2 = (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            View view2 = inflater2.inflate(R.layout.item_menu, null);
            TextView textView2 = (TextView) view2.findViewById(R.id.textView);
            textView2.setText("Elemento " + position);
            textView2.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Toast.makeText(getApplicationContext(), "Elemento " + position + " seleccionado", Toast.LENGTH_SHORT).show();
                }
            });
            return true;
    }
}

```

Al igual que en el menú contextual anterior, el escuchador que se va a utilizar se construirá a partir de onOptionsItemSelected y responderá una u otra cosa, según sea el identificador de la opción pulsada.

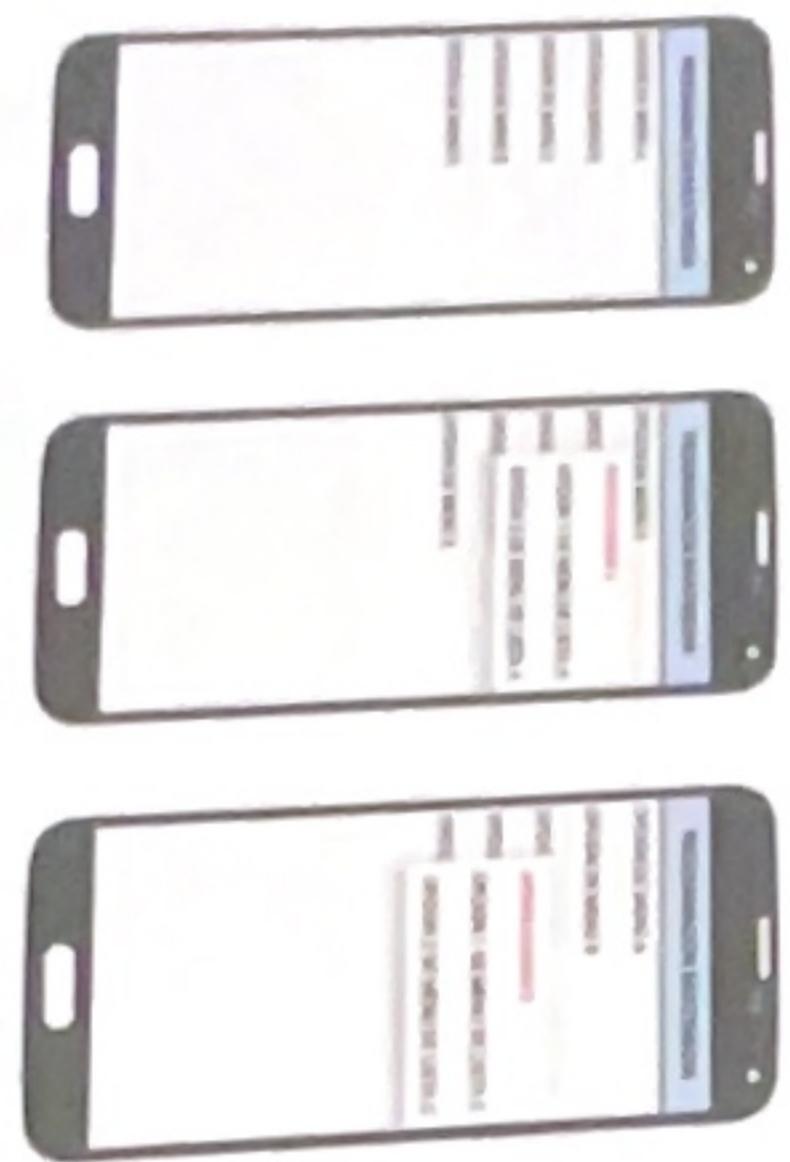


Figura 6.9
Menús contextuales en listas.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cuál de los siguientes escuchadores utilizarías en un OptionsMenu?

- a) onContextItemSelected.
- b) onOptionsMenuSelected.
- c) onItemClickListener.
- d) onItemSelectedListener.

2. ¿Qué método se utiliza para mostrar un menú en pantalla?

- a) onCreateOptionsMenu.
- b) getMenuInflater.
- c) registerForContextMenu.
- d) onCreateContextMenu.

3. ¿Cómo se pueden crear menús con opciones de submenús en Android?

- a) Solo desde XML.
- b) Solo desde Java.
- c) Tanto desde XML como desde Java.
- d) Los submenús no son factibles en Android.

4. En un menú contextual de listado, ¿quién suministra la información del elemento tocado?

- a) El propio elemento.
- b) El objeto listado.
- c) El objeto POJO.
- d) El adaptador.

Como en el apartado anterior, se ha de “inflar” el menú contextual, pero para saber cuáles de ellos debemos “inflar”, obtendremos primero del adaptador el elemento que se ha pulsado y los datos que contiene. Como en el apartado anterior, se ha de “inflar” el menú contextual, pero para saber cuáles de ellos debemos “inflar”, obtendremos primero del adaptador el elemento que se ha pulsado y los datos que contiene.

Los listados son importantes objetos de Android, pertenecientes a ViewGroup, que nos muestran un conjunto de elementos ordenados (con variantes) por los que podemos navegar y seleccionar.

Existen tres variantes principales: ListView (listado clásico), GridView (en forma de rejilla, en el que podemos configurar el número de columnas y el aspecto de estas) y Spinner (aquel cuyos elementos aparecen contraídos y solo se muestra uno antes de ser desplegado). Todos tienen sus escuchadores específicos, que nos devuelven el elemento seleccionado, la posición en la vista y en la lista. Los adaptadores son esenciales para permitir la inclusión de múltiples objetos en los listados, configurar el aspecto y permitir que la estética de los mismos se adapte a la requerida en la aplicación contenadora. De las variantes posibles, dos son las más usadas: los ArrayAdapter (convierte un ArrayList de objetos en vistas) y BaseAdapter (la más usada por su versatilidad y posibilidades). Tanto una como otra necesitan en su construcción un elemento (POJO) que gestione el suministro de datos.

Resumen