ADAPTADORES

Pág. 58 y sgts. del Manual de SGOliver

1. INTRODUCCION

- Android dispone de diversos controles (vistas) que nos permiten seleccionar una opción dentro de una lista de posibilidades. Así, podremos utilizar listas desplegables (***Spinner***) o listas fijas (***ListView***), entre otros.

- Al hablar de los controles de selección, vamos a describir un elemento importante y común a todos ellos, el **adaptador** o **Adapter**.

2. SPINNER

- Es una lista **desplegable** en la que podemos seleccionar una opción.

- Un spinner se define con una entrada del mismo nombre en el archivo de layout, es decir, con una etiqueta **<Spinner>.**

- Si los datos a mostrar en el spinner son estáticos es posible definir la lista de valores como un recurso de tipo **string-array**. Para ello, primero creamos un nuevo fichero XML en la carpeta ***/res/values*** llamado, por ejemplo, ***arrays.xml*** e incluimos en él los valores seleccionables de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planetas">
        <item>Mercurio</item>
        <item>Venus</item>
        <item>Tierra</item>
        <item>Marte</item>
        <item>Júpiter</item>
        <item>Saturno</item>
        <item>Urano</item>
        <item>Neptuno</item>
    </string-array>
</resources>
```

  Habrá una etiqueta **<ítem>** por cada valor que queremos que aparezca en el spinner.

- También es válido añadir el **string-array** como un recurso más en el archivo **strings.xml**.

- Para facilitar la asociación entre estos datos y el control o vista de tipo spinner, podemos utilizar la propiedad **android:entries="@array/nombre_del_string-array":**

```
<!--  Definicion del spinner y enlace con el recurso de tipo array  -->
<Spinner
    android:id="@+id/spinPlanetas"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/planetas" />
```

- Para obtener el contenido del item seleccionado y su posición en el Spinner se pueden emplear los métodos ***getSelectedItem()*** y ***getSelectedItemId().*** Hay que tener en cuenta que la primera posición del spinner tiene el valor 0.

- Podemos probar todo esto mediante el **ejercicio 1**.

- **Eventos**:

  - El más común de los eventos lanzados por la vista Spinner es el generado al seleccionarse una opción de la lista desplegable: evento ***onItemSelected***.

  - Para capturar este evento se procederá de forma similar a lo visto para, por ejemplo, el evento onClick de un botón, mediante el método ***setOnItemSelectedListener()***:

```
spinPlanetas.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        TextView lblResultado = (TextView) findViewById(R.id.lblResultado);
        lblResultado.setText("Elección: " + parent.getItemAtPosition(position) +
                             "\nPosición: " + (position+1));
    }

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

    }
});
```

- Al implementar el listener hay que sobreescribir los métodos ***onItemSelected()*** **y** ***onNothingSelected()*****.**

- El método ***onItemSelected()*** es llamado cuando se selecciona un item.

  Parámetros:
  - parent: El elemento (AdapterView) donde se hizo la selección.
  - view: La vista seleccionada dentro del AdapterView.
  - position: La posición de la vista dentro del Adaptador.
  - id: Cada item puede tener varias filas (en este caso solo tiene una) y este campo indicaría la fila dentro del item.

- El método ***onNothingSelected()*** se llamará cuando no haya ninguna opción seleccionada.

- Para recuperar el dato seleccionado hemos utilizado el método ***getItemAtPosition()*** del parámetro **AdapterView** que recibimos en el evento.

- Podemos probar todo esto mediante el **ejercicio 2**.

Lo que acabamos de ver es la manera más simple de cargar un Spinner: utilizamos únicamente su definición en /res/layout/main.xml y un fichero de recursos en res/values/arrays.xml, con los elementos a mostrar. Pero, si queremos tener el control del contenido del Spinner e incluso modificar el aspecto con el que se muestra, deberemos emplear código Java. Android proporciona para ello un tipo de herramienta que son los **adaptadores**.

3. ADAPTADORES

- El contenido de una lista puede ser estático o variar en función del tiempo o de las acciones del usuario. En este segundo caso, se hace necesario implementar la conexión entre los datos y las vistas que los muestran. Esta conexión es lo que se llama ***Adaptador*** o ***Adapter***.

- Un adaptador es un elemento que hace de intermediario entre una fuente de datos (array, fichero, BBDD) y una interfaz de usuario que muestra esos datos, por ejemplo un **Spinner** o una **ListView**.

- Es el responsable de generar las vistas necesarias para mostrar los datos. Por ejemplo, cada item de una lista puede tener una línea de texto o estar formado por varios subcomponentes (texto e imagen…). En cada caso, es el adaptador el encargado de representar eso.

- Android proporciona por defecto varios tipos de adaptadores sencillos.

- El adaptador más sencillo es el ***ArrayAdapter***, que proporciona datos a un Spinner o a una ListView a partir de un array de objetos de cualquier tipo.

Documentación en http://developer.android.com/reference/android/widget/ArrayAdapter.html

3.1 ADAPTADOR CON ARRAY ESTATICO

- Los elementos que forman un spinner también pueden ser definidos **mediante código Java**. El ejemplo equivalente al anterior sería:

```
private String[] arrayPlanetas =
{"Mercurio","Venus","Tierra","Marte","Júpiter","Saturno","Urano","Neptuno"};
```

- Para rellenar el spinner crearemos un adaptador de tipo ***ArrayAdapter***.

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

- La clase ***ArrayAdapter*** tiene varios constructores:

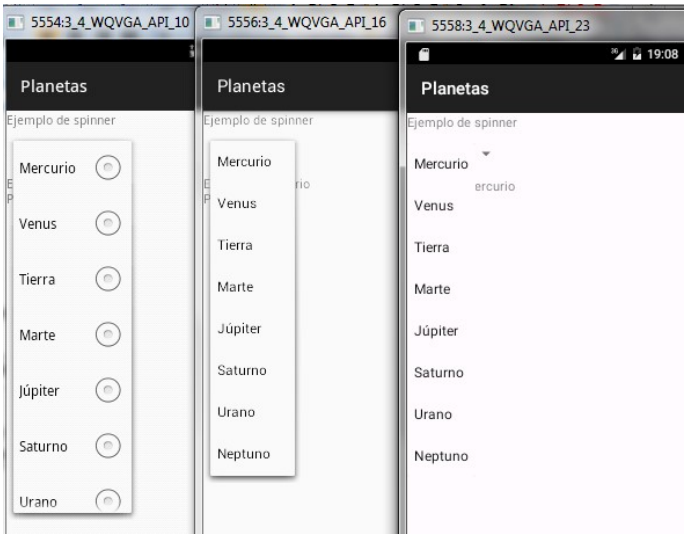
Public constructors
`ArrayAdapter(Context context, int resource)` Constructor
`ArrayAdapter(Context context, int resource, int textViewResourceId)` Constructor
`ArrayAdapter(Context context, int resource, T[] objects)` Constructor.
`ArrayAdapter(Context context, int resource, int textViewResourceId, T[] objects)` Constructor.
`ArrayAdapter(Context context, int resource, List<T> objects)` Constructor
`ArrayAdapter(Context context, int resource, int textViewResourceId, List<T> objects)` Constructor

- En este ejemplo utilizamos el constructor con tres parámetros:
  - Un objeto Context, que referencia a la propia actividad (***this***).
  - La manera en que será mostrado el spinner. Android proporciona una serie de Layouts predefinidos de los cuales los más habituales son:
    - ***simple_spinner_item***
    - ***simple_spinner_dropdown_item***
  - El array de objetos (Strings) que hemos creado (**arrayPlanetas**).
- Como último paso, debemos asignar el adaptador al spinner. Para esto, necesitamos un objeto de la clase **Spinner** que apunte al spinner que hemos declarado en el layout, y le asignamos el adaptador mediante el método ***setAdapter()***:

```
spinPlanetas.setAdapter(adaptador);
```

- Podemos probar todo esto mediante el **ejercicio 3.**
- Los layouts predefinidos se muestran con aspecto diferente según el nivel de API.

Aspecto que produce en diferentes APIs



3.2 ADAPTADOR CON ARRAY DE RECURSOS XML

- Para rellenar el spinner crearemos un adaptador de tipo ***ArrayAdapter:***

```
ArrayAdapter<CharSequence> adaptador = ArrayAdapter.createFromResource
(this, R.array.planetas, android.R.layout.simple_spinner_item);
```

- El método ***createFromResource()*** crea un objeto de tipo ArrayAdapter desde un recurso externo y recibe tres parámetros:
  - El contexto, normalmente el actual al que haremos referencia con 'this'.
  - El identificador del recurso XML que contiene los items que vamos a mostrar en el spinner.
  - La manera en que será mostrada la lista.

static ArrayAdapter<CharSequence>	createFromResource(Context context, int textArrayResId, int textViewResId) Creates a new ArrayAdapter from external resources.

- Podemos probar todo esto mediante el **ejercicio 4.**

3.3 ADAPTADOR CON ARRAY DINAMICO

- Con lo visto hasta ahora, el contenido del Spinner era estático y se definía en tiempo de compilación.
- Si usamos arrays dinámicos podemos crear la fuente de los datos en tiempo de ejecución antes de pasársela al adaptador. De esta forma, podemos obtener datos de ficheros, bases de datos, etc y crear una fuente de datos para un Spinner en tiempo de ejecución.
- El manejo del adaptador es similar al visto para arrays estáticos (apartado 3.1)
- Podemos probar esto mediante el **ejercicio 5.**

4. LISTVIEW

- Es una vista que permite al usuario seleccionar un elemento.
- Se diferencia de un Spinner en que muestra en pantalla todos sus elementos, es decir, no se trata de una lista emergente.
- Al igual que con los Spinner, usaremos una fuente de datos (array estático, array dinámico, recurso xml, etc.), crearemos posteriormente el adaptador de tipo **ArrayAdapter** y asignaremos este adaptador al control de tipo **ListView** mediante el método ***setAdapter()***.
- Para añadir una vista de tipo ListView a la interfaz de usuario se puede emplear el código:

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

- Al igual que la vista Spinner, ListView también acepta la **propiedad *android:entries***
- También, al igual que lo visto para los spinner, podemos crear un adaptador para cargar los datos en la ListView. Para ello se puede emplear un layout genérico de Android para los controles de tipo ListView (***android.R.layout.simple_list_item_1***), formado únicamente por un TextView.

```
ArrayAdapter<CharSequence> adaptador = ArrayAdapter.createFromResource
    (this, R.array.planetas, android.R.layout.simple_list_item_1);
```

- El ejemplo con los nombres de los planetas se visualizaría como muestra la imagen siguiente:

- Podemos probar esto mediante el **ejercicio 11 (Planetas6).**

- **Eventos**: Si queremos realizar cualquier acción al pulsar sobre un elemento de la lista tendremos que implementar el evento ***onItemClick*.**

```
lvPlanetas.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
        Toast.makeText(MainActivity.this, "Elección: " +
                parent.getItemAtPosition(pos), Toast.LENGTH_LONG).show();
    }
});
```

- En los casos de un array estático definido en el código java o de arrays dinámicos, se hace de forma similar a lo visto en los controles tipo Spinner.

5. LISTAS PERSONALIZADAS

- La vista ***ListView*** estándar sólo muestra una lista de elementos (lista de strings).

- Podemos personalizar el aspecto de una listView al modificar el layout que se utilizará para visualizar cada elemento, en lugar de aplicar un layout por defecto.

EJEMPLO1

- Podemos probar esto mediante el **ejercicio 12 (ListasPersonalizadas).**

- Añadiremos en cada fila una imagen (la misma imagen en todas las filas, en este caso). Por ejemplo, podemos aprovechar el archivo "*ic_launcher.png"* que viene incluido por defecto en los recursos de cualquier aplicación. El resultado es:

- Para ello, creamos un nuevo archivo .xml en la carpeta *res/layout*, para definir el **layout de cada fila** de la nueva lista. Por ejemplo, para el caso anterior:

```
<!-- layout de cada fila de la lista  -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ImageView
        android:id="@+id/imgPlaneta"
        android:layout_width="22dp"
        android:layout_height="22dp"
        android:layout_margin="6dp"
        android:src="@drawable/ic_launcher">
    </ImageView>

    <TextView
        android:id="@+id/tvPlaneta"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="21sp">
    </TextView>
</LinearLayout>
```

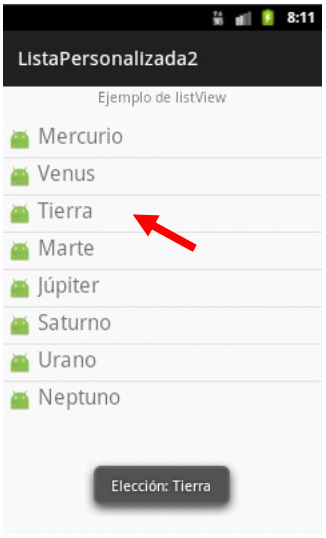
- En la actividad principal, indicamos al constructor del adaptador que queremos utilizar nuestro propio fichero de layout: "*cadafila.xml*", para especificar como queremos mostrar los elementos de la lista, y que en cada fila el texto se mostrará en la vista cuyo identificador es "*tvPlaneta*" (el TextView en nuestro layout). Es decir, empleamos el constructor que tiene la siguiente estructura:

```
ArrayAdapter(Context context, int resource, int textViewResourceId, T[]
objects)

Constructor.
```

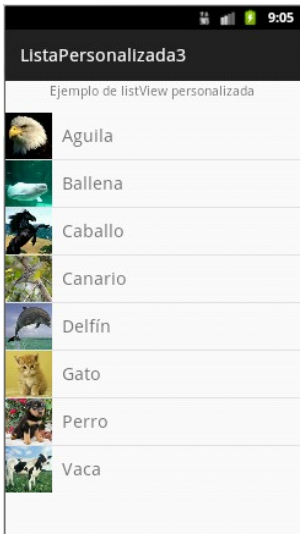
```
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
        R.layout.cadafila, R.id.tvPlaneta, arrayPlanetas);
```

- Para realizar cualquier acción al pulsar sobre un elemento de la lista podemos implementar el evento ***onItemClick***, como vimos anteriormente:



EJEMPLO 2

- Ahora vamos a colocar diferentes imágenes en cada fila. Por ejemplo (podemos probar esto mediante el **ejercicio 13 (ListasPersonalizadas2)**:



- En primer lugar, creamos un nuevo archivo de layout ("***layout_fila.xml***") con el diseño que queremos para cada item de nuestra ListView. Por ejemplo:

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

    <ImageView
        android:id="@+id/imgAnimal"
        android:layout_width="50dp"
        android:layout_height="50dp" />

    <TextView
        android:id="@+id/tvAnimal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:layout_gravity="center_vertical"
        android:textSize="20dp"/>
</LinearLayout>
```

- Luego vamos a crear el contenido de la lista. Para ello, añadimos las imágenes a la carpeta ***res/drawable***.

- Podemos crear una clase modelo (por ejemplo, clase "Animal") y después un array de objetos de esa nueva clase, con los valores que nos interesen. Pero también podemos hacerlo directamente mediante dos arrays definidos en el archivo principal:

```
arrayAnimales = new String[]{"Aguila", "Ballena", "Caballo", "Canario",…};
arrayIdFotos = new Integer[]{R.drawable.aguila, R.drawable.ballena,
                             R.drawable.caballo, R.drawable.canario,… };
```

- Como las imágenes están almacenadas en la carpeta "drawable" podemos acceder a cada una de ellas mediante su Id (de tipo entero). Y posteriormente podremos asignar cada imagen a una vista de tipo ***ImageView*** con este valor, empleando el método ***setImageResource()***.

void	setImageResource(int resId) Sets a drawable as the content of this ImageView.

- A continuación, creamos nuestro **adaptador personalizado**. Para ello, añadiremos una nueva clase Java al proyecto, a la que llamaremos por ejemplo "***AdaptadorPersonalizado***" y que hereda de la clase ***ArrayAdapter***, la cual ya implementa los métodos que nosotros necesitamos, y sólo tendremos que sobreescribir uno de ellos.

```
public class AdaptadorPersonalizado extends ArrayAdapter{…}
```

- En esta clase necesitamos crear tres variables: una para el contexto y dos para los datos: los nombres de los animales y las imágenes.

```
private Activity context;
private String[] animales;
private Integer[] fotosIds;
```

- Estas variables las inicializamos en el constructor, en donde hacemos también la llamada al constructor de la clase padre (en este caso, *ArrayAdapter*) mediante ***super()***.

- Como ya habíamos visto, la clase ArrayAdapter tiene varios constructores. En este caso, estamos empleando el constructor de tres parámetros: el **contexto**, la referencia al **layout** con el diseño personalizado de nuestras filas, y el array con los **datos**.

```
public class AdaptadorPersonalizado extends ArrayAdapter<String>{
    private Activity context;
    private String[] animales;
    private Integer[] fotosIds;

    // constructor
    public AdaptadorPersonalizado(Activity context, String[] animales,
                                  Integer[] fotosIds) {
        super(context, R.layout.layout_fila, animales);
        this.context = context;
        this.animales = animales;
        this.fotosIds = fotosIds;
    }
(…)
```

- Utilizaremos el contexto para "inflar" las vistas que mostraremos en cada fila de la ListView. Y eso se hace en el **método más importante de esta clase**, que es el método ***getView()***. Este método es llamado cada vez que hay que "pintar" un item de la ListView en la pantalla del dispositivo.

View	getView(int position, View convertView, ViewGroup parent) Get a View that displays the data at the specified position in the data set.

```
@Override
public View getView(int position, View view, ViewGroup parent){(…)}
```

- Simplificando, podemos decir que se llama "inflar" a generar objetos Java a partir de elementos XML.

- El método ***getView()*** es el que realmente se encarga de inflar y mostrar las vistas de cada fila de la listView. Es un método implementado en la clase base ***ArrayAdapter***, que debemos sobreescribir para adaptarlo a nuestras necesidades:

- Lo primero es "inflar" una nueva vista ("**fila**") que será la que mostrará cada item de la ListView. Para ello creamos un objeto de tipo ***LayoutInflater***, y llamamos a su método ***inflate()***.

LayoutInflater

added in API level 1 ☆☆☆☆☆

```
public abstract class LayoutInflater
extends Object
```

java.lang.Object
↳ android.view.LayoutInflater

Instantiates a layout XML file into its corresponding `View` objects. It is never used directly. Instead, use `Activity.getLayoutInflater()` or `Context.getSystemService(Class)` to retrieve a standard LayoutInflater instance that is already hooked up to the current context and correctly configured for the device you are running on.

(...)

`View`	`inflate(int resource, ViewGroup root)` Inflate a new view hierarchy from the specified xml resource.

```
LayoutInflater inflater = context.getLayoutInflater();
View fila = inflater.inflate(R.layout.layout_fila, null);
```

Para inicializar el objeto "***inflater***" hemos utilizado el contexto ("***context***") que recibimos desde el constructor. Es muy importante que el contexto recibido sea realmente el de la Activity que maneja el ListView.

- A partir de esta vista ***("fila")***, recogeremos los controles que contiene para poder manipularlos. O, dicho de otro modo, vamos a obtener una referencia a todas las vistas de nuestro layout:

```
TextView tvAnimal=(TextView) fila.findViewById(R.id.tvAnimal);
ImageView imagen=(ImageView) fila.findViewById(R.id.imgAnimal);
```

En este caso, en lugar de poner el método ***findViewById()*** directamente, lo hemos llamado a partir del objeto View que creamos ("***fila***"). Esto es así porque este método no se encuentra en la clase ***ArrayAdapter*** y, además, porque el TextView y el ImageView a los que queremos acceder realmente se encuentran dentro de esa vista.

- Rellenamos el contenido de cada fila según su posición, mediante el parámetro "***position***" del método ***getView()***. El valor de "position" se refiere al número de fila que estamos inflando, y éste siempre deberá coincidir con el índice del array que contiene nuestros datos. Es decir, en la fila 0 tendremos el animal que ocupa la posición 0 del array de animales, y así sucesivamente.

```
tvAnimal.setText(animales[position]);
imagen.setImageResource(fotosIds[position]);
```

Hemos asignado la imagen mediante el método ***setImageResource()*** de la clase ***ImageView,*** pasándole como parámetro la referencia al objeto que queremos utilizar, es decir, el ID que almacenamos en el array "**fotosIds**".

- Por último, retornamos nuestro elemento de tipo View:

```
return fila;
```

- El método al completo quedaría:

```
@Override
public View getView(int position, View view, ViewGroup parent){
    LayoutInflater inflater=context.getLayoutInflater();
    View fila = inflater.inflate(R.layout.layout_fila, null);

    TextView tvAnimal=(TextView) fila.findViewById(R.id.tvAnimal);
    ImageView imagen=(ImageView) fila.findViewById(R.id.imgAnimal);

    tvAnimal.setText(animales[position]);
    imagen.setImageResource(fotosIds[position]);

    return fila;
}
```

- Una vez que hemos creado nuestro adaptador personalizado, en la actividad que contiene el ListView debemos crear un objeto de este tipo y asignárselo al objeto ListView, de forma similar a como hicimos para el caso de un adaptador simple como ArrayAdapter:

```
AdaptadorPersonalizado adaptador = new AdaptadorPersonalizado(this,
arrayAnimales, arrayIdFotos);
lvAnimales.setAdapter(adaptador);
```