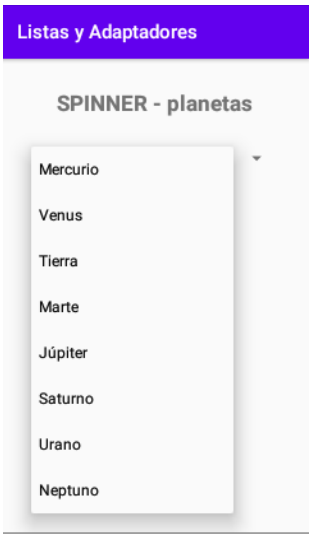
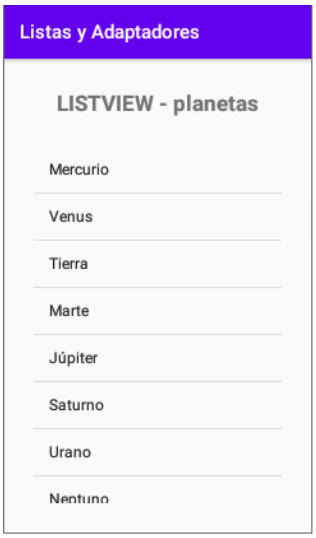
LISTADOS Y ADAPTADORES

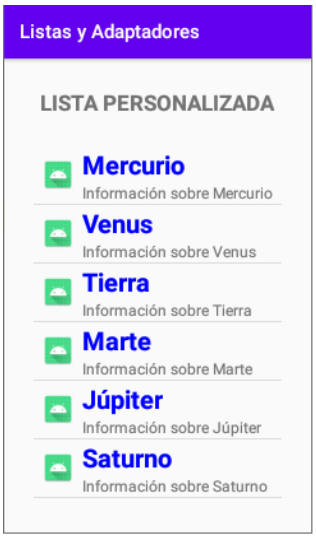
1. INTRODUCCION

- Android dispone de diversas 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.





- Como siempre, elaborar y gestionar una lista simple (***ListView*** o ***Spinner***) es relativamente sencillo. Pero en ocasiones, según las necesidades de nuestra app, necesitaremos personalizar el aspecto de cada elemento del listado:



- Para esto, la programación se hace más compleja y requiere el uso de un elemento adicional: el **adaptador** o **Adapter**.

- El adaptador es el elemento que sirve de conexión entre los datos que queremos visualizar y las vistas que los muestran

(Imagen de CABRERA RODRIGUEZ, J: "**Programación multimedia y dispositivos móviles**". Ed. Síntesis)

- La imagen muestra como 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 sólo una línea de texto (como en las capturas iniciales) o estar formado por varios subcomponentes (por ejemplo, textos e imagen, en la lista personalizada anterior). 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.

2. SPINNER

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

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

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

- Si los datos que queremos 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
<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>
```

Hay 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** pero, si lo hacemos así, quedaría peor estructurado.

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

```xml
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(),*** de la clase ***AdapterView***. Por ejemplo:

```
String planeta=spPlanetas.getSelectedItem().toString();
```

- La primera posición del spinner tiene el valor 0.

- **El evento más común lanzado por la vista Spinner es el que se produce cuando seleccionamos una opción de la lista desplegable: el evento *onItemSelected***

- Este evento está asociado a la interfaz ***OnItemSelectedListener***

- Para realizar un escuchador o **listener** que responda al evento en cuestión se procederá de forma similar a lo visto para, por ejemplo, el evento onClick de un botón. Es decir, emplearíamos el método ***setOnItemSelectedListener()***:

```
//escuchador del spinner
spPlanetas.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
        //operaciones pertinentes
    }

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

    }
});
```

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

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

  Parámetros principales:
  - *adapterView*: El elemento (adaptador) donde se hizo la selección.
  - *view*: La vista seleccionada dentro del AdapterView.
  - *i*: La posición de la vista dentro del adaptador (posición inicial=0).

- El método ***onNothingSelected()*** no tiene utilidad para nosotros.

- También podemos recuperar el elemento seleccionado utilizando el método ***getItemAtPosition()*** del parámetro ***adapterView*** que recibimos en el evento.

```
@Override
public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l)
{
    String eleccion1=spPlanetas.getSelectedItem().toString();
    String eleccion2=adapterView.getItemAtPosition(i).toString();
    //resto de operaciones
}
```

- Podemos probar todo esto para obtener el ejemplo recogido en la primera imagen capturada.

- Lo que acabamos de ver es la manera más simple de gestionar 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, debemos aprender más sobre los adaptadores.

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

3. ADAPTADORES

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:

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

- Los pasos siguientes son **crear el adaptador** que reconozca y pueda darles forma a dichos datos, y **asignar el adaptador al spinner,** mediante el método ***setAdapter()***:

```
//creamos el elemento adaptador
ArrayAdapter<String> adaptador = new ArrayAdapter<String>
        (this, android.R.layout.simple_spinner_item, arrayPlanetas);
//asignamos al Spinner el adaptador con los datos ya cargados
spPlanetas.setAdapter(adaptador);
```

- 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 hemos utilizado el constructor con tres parámetros:
  - Un objeto de tipo 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 queremos visualizar.
- Los layouts predefinidos, como "*simple_spinner_item*", se pueden visualizar con aspecto diferente según el nivel de API.
- Podemos probar todo esto en un ejemplo similar al anterior en cuanto a funcionalidad, pero con este nuevo código que incluye el uso de un adaptador simple**.**

3.2 ADAPTADOR CON ARRAY DE RECURSOS XML

- También es posible tener los datos como un recurso, pero cargarlos en el spinner por medio de un adaptador, como acabamos de ver.
- En este caso, podemos hacer uso del método ***createFromResource()***

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

- El método ***createFromResource()*** recibe tres parámetros:
  - El contexto, normalmente el actual al que haremos referencia con 'this'.
  - El identificador del recurso XML que contiene los datos 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.

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

- La funcionalidad sería la misma si recuperamos el recurso mediante el método ***getResources().getStringArray()***, y lo asignamos a un array declarado en el código. Este array es el que pasaríamos después al adaptador.
- Podemos probar todo esto en un ejemplo similar al anterior en cuanto a funcionalidad, pero con este nuevo código que incluye la carga del adaptador desde un recurso definido en XML**.**

3.3 ADAPTADOR CON ARRAY DINAMICO

- Con lo visto hasta ahora, el contenido del Spinner era siempre 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.

4. LISTVIEW

- Se diferencia de un Spinner en que muestra en pantalla todos sus elementos, es decir, no se trata de una lista emergente.

- Para añadir una vista de tipo ListView a la interfaz de usuario se puede emplear el código:

```
<ListView
    android:id="@+id/lvPlanetas1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

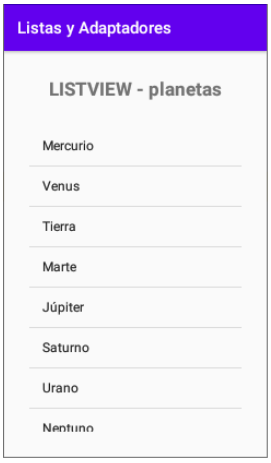
- Igual que ocurría con Spinner, ListView también acepta la **propiedad** ***android:entries***

- El **evento** principal asociado a una ListView es el que se produce cuando seleccionamos una opción. Se trata del evento ***onItemClick()***

- El **listener** que responde al evento en cuestión se implementa igual que en ocasiones anteriores; es decir, con el método ***setOnItemClickListener()***:

```
lvPlanetas.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        //obtenemos el item seleccionado
        String seleccion=adapterView.getItemAtPosition(i).toString();
        //código siguiente
    }
});
```

- Podemos probar esto elaborando una lista como la que se recoge en la segunda captura de este mismo documento, la cual vemos de nuevo aquí**:**



- 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 las vistas 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);
```

- También, como en los spinner, podemos recuperar el recuso directamente con el método ***createFromResource()*** o bien mediante ***getResources().getStringArray()***.

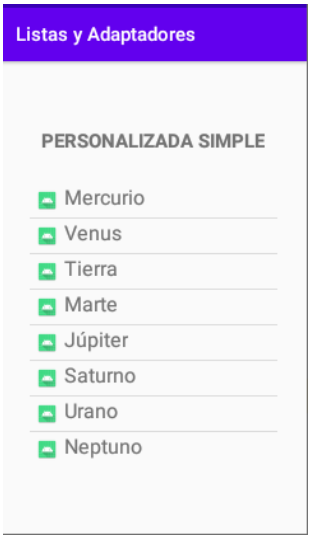
- En los casos de un array estático definido en el código java o de arrays dinámicos, se hace de forma similar a lo visto en los controles tipo Spinner.

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, como hicimos hasta ahora.

- Por ejemplo, la lista que se muestra en la captura siguiente tiene una imagen y un texto en cada ítem. En este caso simple, todos los ítems tienen la misma imagen:



- Para ello, creamos nuestro propio archivo de layout para cada ítem. Es el layout que vamos a utilizar en lugar del layout defectivo de Android. Por ejemplo:

```
<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, creamos el adaptador utilizando **otro** de sus constructores:

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

Constructor.

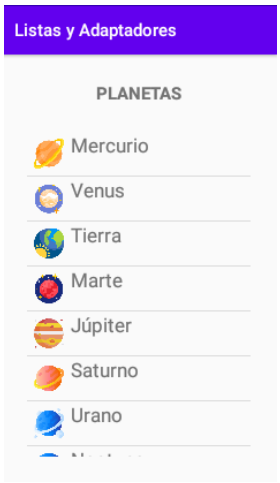
- Tiene cuatro parámetros:
  - El contexto
  - El id de nuestro layout (el que nosotros hemos creado para cada ítem)
  - El id de la TextView en donde vamos a visualizar cada texto
  - El conjunto de elementos

```
ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,
        R.layout.fila_simple,
        R.id.tvPlaneta,
        getResources().getStringArray(R.array.planetas));
```

- Esta forma de implementar la lista es fácil pero de nuevo nos limita mucho a la hora de personalizar nuestras listas.

- Si deseamos tener el control total sobre el contenido de cada ítem tenemos que implementar nuestro propio adaptador.

- Vamos a probar esto con un ejemplo similar, pero con imágenes diferentes para cada ítem de la lista:



- En primer lugar, lo veremos de la forma que implica menos cambios respecto a los algoritmos anteriores. Luego implementaremos una versión mejorada.

- Pasos a seguir:

  - Crear el archivo de **layout para cada fila**.
    En este caso, nos vale el del apartado anterior, pero le eliminaremos la propiedad **android:src** puesto que ahora no tenemos una única imagen.

  - Añadir las **imágenes** como recursos de tipo "**drawable**". Esto nos permitirá acceder a cada una de ellas mediante su id (de tipo entero). Y, posteriormente, a partir de este valor, podremos asignar cada imagen a una vista de tipo ***ImageView***, empleando el método ***setImageResource()***.

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

  - **Crear** nuestro **adaptador personalizado**. Para ello, añadiremos al proyecto una nueva clase Java que hereda de la clase ***ArrayAdapter***. Esta clase **ArrayAdapter** dispone de los métodos que nosotros necesitamos, pero tendremos que sobreescribir uno de ellos, el más importante en este caso: método ***getView()***

```
public class AdaptadorPersonalizado extends ArrayAdapter {
        public AdaptadorPersonalizado(@NonNull Activity context,
                                      int resource,
                                      @NonNull Object[] objects) {
                super(context, resource, objects);
        }
```

    Tal y como lo vamos a hacer, **es muy importante que el contexto sea la propia Activity que maneja el ListView**

  - El método ***super()*** realiza una llamada al **constructor de tres parámetros** de la clase padre: ***super****(context, resource, objects);*

  - Es el mismo constructor que usamos en anteriores ocasiones:
    - **contexto**,
    - id del **layout** que hemos creado para dar formato a cada fila y
    - **array de objetos** (en nuestro caso, strings con el nombre de los planetas)

  - Los restantes datos que formarán parte de nuestra lista, se los pasaremos en el constructor de nuestro adaptador (para eso estamos personalizándolo según nuestras necesidades).

  - El aspecto, hasta ahora, sería:

```
private Activity context;
private String[] arrayPlanetas;
private int[] arrayIdFotos;

public AdaptadorPersonalizado (@NonNull Activity context,
                               int miLayout,
                               @NonNull String[] arrayPlanetas,
                               int[] arrayIdFotos)
{
    super(context, miLayout, arrayPlanetas);//constructor defectivo
```

```
        this.context=context;
        this.arrayPlanetas=arrayPlanetas;
        this.arrayIdFotos=arrayIdFotos;
    (…)
    }//end adaptador personalizado
```

- Este tercer parámetro nos permitirá, en sucesivas mejoras, pasar al adaptador el **array de objetos** creados a partir de una clase (POJO) según las características de cada proyecto. Por ejemplo, en este caso, podríamos crear una clase ***Planeta***, con dos elementos (imagen y texto) y los correspondientes métodos getter y setter.

- Una vez que hemos creado nuestro adaptador personalizado, creamos una instancia del mismo en la Activity que contiene la ListView. Como siempre, debemos asignar dicho adaptador a la lista e implementar el escuchador.

```
//crear una instancia de mi adaptador
AdaptadorPersonalizado adaptador=new AdaptadorPersonalizado(this,
                R.layout.fila_imagen_planeta, arrayPlanetas, arrayIdFotos);
//asignar a la lista
lvPlanetas.setAdapter(adaptador);
```

- Dentro de la clase donde hemos creado nuestro adaptador personalizado aún tenemos que **sobreescribir el método *getView()***, el cual se encargará de generar ("inflar"), y rellenar con los datos correspondientes, cada uno de los ítems de la ListView.

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

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

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

- Este método será llamado cada vez que haya que mostrar un nuevo elemento (ítem) en la lista y para ello se realiza el siguiente proceso:

    - Primero se "infla" el layout que da formato a cada elemento de la lista.
    - Para esto, instanciamos un objeto de tipo ***LayoutInflater*** haciendo uso del método estático ***getLayoutInflater()*** , 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.

(...)

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

- A partir de esta vista (View **fila**) referenciamos los elementos que aparecen en nuestro layout de fila, que son los que acabarán "rellenándose" con cada uno de los datos que se encuentran en la posición del array que se va a mostrar en esa llamada al método getView().

```
TextView tvPlaneta = fila.findViewById(R.id.tvPlaneta);
ImageView imgPlaneta = fila.findViewById(R.id.imgPlaneta);
/* insertamos en cada componente su valor (planeta/imagen) según su pos */
tvPlaneta.setText(arrayPlanetas[position]);
imgPlaneta.setImageResource(arrayIdFotos[position]);
```

- Como se ve en el código, 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, el TextView y el ImageView a los que queremos acceder realmente se encuentran dentro de esa vista.

- También se ve que hemos rellenado 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 texto que ocupa la posición 0 del array de strings, y así sucesivamente.

- Para finalizar, el método retorna la View creada e inflada, en nuestro caso, llamada "fila":

```
return fila;
```

6. VERSIONES MEJORADAS

- Como ya hemos comentado en ejemplos anteriores, incluir en la clase principal fuentes de datos con mucha información no es lo más conveniente.

  En este ejemplo, sería más correcto sustituir el código con el que inicializamos el array de los ids de las imágenes por una llamada al método getResources():

  Es decir, en lugar del código

```
int[] arrayIdFotos={R.drawable.mercurio, R.drawable.venus, R.drawable.tierra,
                    R.drawable.marte, R.drawable.jupiter, R.drawable.saturno,
                    R.drawable.urano, R.drawable.neptuno};
```

  podríamos haber usado:

```
arrayIdFotos=getResources().obtainTypedArray(R.array.fotos);
```

  En este caso, **arrayIdFotos** debería ser un objeto de la clase ***TypedArray*** , lo que nos permititía "rellenar" la imagen en el método ***getView()*** con el método ***setImageDrawable()*** como se muestra en las siguientes líneas:

```
ImageView imgPlaneta = fila.findViewById(R.id.imgPlaneta);
Drawable drawable=arrayIdFotos.getDrawable(position);
imgPlaneta.setImageDrawable(drawable);
```

void	setImageDrawable(Drawable drawable) Sets a drawable as the content of this ImageView.

- Sería también una mejora en el código el implementar una **clase POJO**, con los atributos correspondientes a los datos que nos interesen así como los métodos get y set.

  En este caso, podríamos crear una clase ***Planeta***, con dos atributos (nombre e imagen) y cargar los datos de nuestros planetas en un **array de objetos de tipo Planeta**, que sería el parámetro pasado en tercera posición al constructor del ArrayAdapter.

  De esta forma, podemos utilizar un ArrayAdaptar "tipado": ***ArrayAdapter<Planeta>***

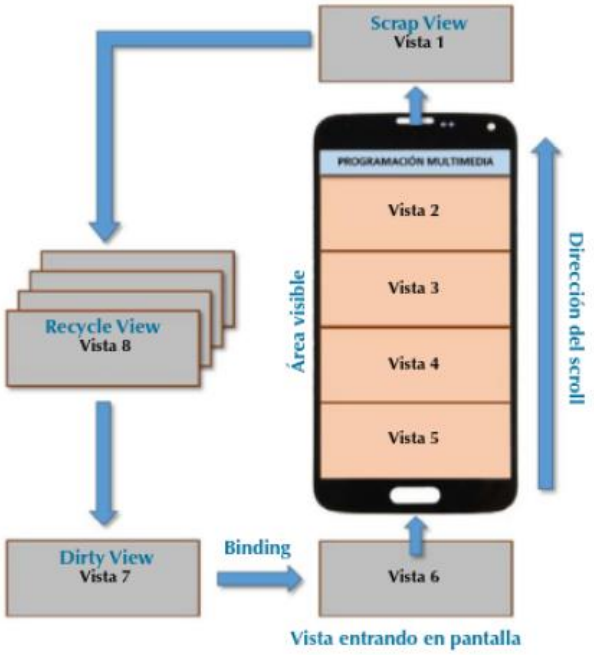
- **Optimización en el uso de recursos**

  Aún podemos mejorar más la eficiencia de nuestro código de forma que la app ahorre trabajo de CPU y uso de recursos como batería y memoria, "reciclando" las vistas ya utilizadas.

  Cuando se enlaza un ListView a un adaptador, éste se encarga de crear las instancias de las filas necesarias hasta que haya suficientes elementos para

completar la altura definida para la lista, no siendo necesario entonces preparar elementos adicionales en memoria, elementos que no se visualizan porque no caben en el área visible de la pantalla.

Pero, si se genera un scroll vertical, los elementos que salen de la pantalla, por arriba o por abajo, quedarán en memoria para un uso posterior, y se incorporan tantos ítems como los que han sido guardados en memoria. Es decir, las vistas ya creadas y que abandonan temporalmente el área visible, pueden ser reutilizadas sin necesidad de tener que "inflarse" de nuevo, operación que consume muchos recursos.



(Imagen de CABRERA RODRIGUEZ, J: "**Programación multimedia y dispositivos móviles**". Ed. Síntesis)

Esta idea nos lleva a implementar un código más eficiente de forma que la aplicación ahorre trabajo de CPU y uso de recursos como batería y memoria.

Este código está explicado a partir de la pág. 67 en el **manual de SGOliver**, también accesible en el enlace siguiente (que forma parte del curso indicado en el Aula Virtual)

https://www.sgoliver.net/blog/interfaz-de-usuario-en-android-controles-de-seleccion-iii/

Se resume en lo siguiente:

1. **Reutilizar los layouts "desaparecidos"**:

   Siempre que exista algún layout que pueda ser reutilizado, éste se va a recibir a través del parámetro ***convertView*** del método ***getView()***. De esta forma, en los casos en que este parámetro no sea null podremos obviar el trabajo de inflar el layout. El código pasaría a ser:

```
@Override
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent){
    // primera optimización
    View fila=convertView;
    if (fila == null) {
        LayoutInflater layoutInflater = context.getLayoutInflater();
        fila = layoutInflater.inflate(R.layout.fila_imagen_planeta, null);
    }
```

2. **Guardar la referencia a los elementos del layout**:

   Aprovechar que estamos "guardando" un layout que no sirve para guardar también la referencia a los elementos que lo componen, de tal forma que no haya que volver a buscarlos mediante el ***findViewById()***.

   Esto se suele hacer con lo que se llama un "**patrón ViewHolder**":

   - Creamos una clase ViewHolder que sólo va a contener una referencia a cada uno de los elementos que tengamos que manipular en nuestro layout. En nuestro ejemplo sería:

```
private static class ViewHolder{
    TextView tvPlaneta;
    ImageView imgPlaneta;
}
```

   - Inicializar el objeto ViewHolder la primera vez que inflemos un elemento de la lista y asociarlo a dicho elemento para que podamos recuperarlo posteriormente.

```
@Override
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent)
{
    View fila=convertView;
    ViewHolder viewHolder;

    if (fila == null) {
        LayoutInflater layoutInflater = context.getLayoutInflater();
        fila = layoutInflater.inflate(R.layout.fila_imagen_planeta, null);

        viewHolder=new ViewHolder();
         // resto de código
      }
```

   - Dicha asociación la hacemos aprovechando que todos los elementos tienen una propiedad llamada ***Tag***, que podemos asignar y recuperar con los métodos ***setTag()*** y ***getTag()***, respectivamente.
   - Cuando el parámetro ***convertView*** no sea nulo (rama *else*), sabremos que tenemos disponibles las referencias a sus elementos hijos a través de su propiedad Tag

- Nuestro ejemplo:

```
if (fila == null) {
    LayoutInflater layoutInflater = context.getLayoutInflater();
    fila = layoutInflater.inflate(R.layout.fila_imagen_planeta, null);

    viewHolder=new ViewHolder();
    viewHolder.tvPlaneta=fila.findViewById(R.id.tvPlaneta);
    viewHolder.imgPlaneta=fila.findViewById(R.id.imgPlaneta);
    fila.setTag(viewHolder);
}
else{
    viewHolder=(ViewHolder)fila.getTag
}
```

- Finalmente, asignamos cada valor a su correspondiente elemento del layout, pero lo hacemos desde el objeto vievHolder:

```
viewHolder.tvPlaneta.setText(arrayPlanetas[position]);
…
viewHolder.imgPlaneta.setImageDrawable(drawable);
```