



## Ejercicio 1: P\_26\_Intent\_Seleccion

1. La aplicación tiene tres actividades:
  - a. MainActivity: basada en plantilla EmptyActivity
  - b. SegundaActivity: basada en plantilla BasicActivity y con "padre jerárquico" MainActivity
  - c. TerceraActivity: basada en plantilla BasicActivity y con "padre jerárquico" MainActivity



2. Usa las diversas soluciones que hay para construir Adapter
3. En la tercera actividad, observa el comportamiento distinto de  y 

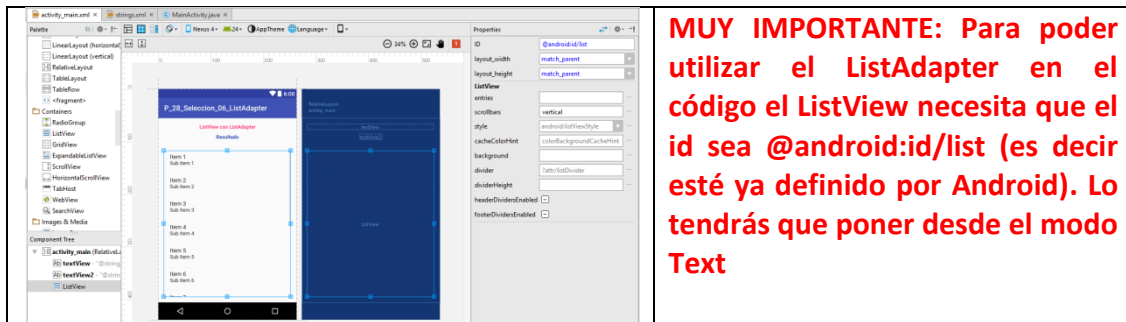
## Ejercicio 2: P\_27\_Intent\_Seleccion\_02

1. La aplicación tiene dos actividades, basadas ambas en la plantilla Empty. Desde MainActivity se lanza SegundaActivity y desde esta se vuelve a MainActivity (recuerda startActivityForResult)



### Ejercicio 3: Proyecto P\_28\_Seleccion\_06\_ListAdapter (Manejando ListAdapter)

1. Crea un nuevo proyecto con MainActivity basada en un Empty Activity (ya trabajaremos más adelante con ActionBar y botones flotantes!)
2. El layout de la actividad principal debe ser semejante al siguiente:



**MUY IMPORTANTE:** Para poder utilizar el ListAdapter en el código el ListView necesita que el id sea **@android:id/list** (es decir esté ya definido por Android). Lo tendrás que poner desde el modo Text

3. Crea en el fichero res/values/strings.xml los arrays de opciones y cursos.
4. Modifica el fichero MainActivity.java para que su contenido sea (con los import necesarios) el siguiente e interpreta su contenido (en rojo lo nuevo!):

```
public class MainActivity extends ListActivity {
```

```
    private TextView textView;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
        textView = (TextView) findViewById(R.id.textview2);
```

```
        ListView listView = (ListView) findViewById(android.R.id.list);
```

```
        String[] asignaturas= getResources().getStringArray(R.array.opciones);
```

```
        ListAdapter adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, asignaturas);
```

```
        listView.setAdapter(adapter);
```

```
    }
```

```
    /*
```

Método encargado de detectar una pulsación sobre un elemento del ListActivity. Los parámetros son:

l: El ListView donde ocurrió el click

v: La vista (View) que fue pulsada

position: La posición de la vista en la lista

id: El identificador de la fila del ítem que fue pulsado

```
    */
```

```
    @Override
```

```
    protected void onListItemClick(ListView l, View v, int position, long id) {
```

```
        super.onListItemClick(l, v, position, id);
```

```
        String[] cursos= getResources().getStringArray(R.array.cursos);
```

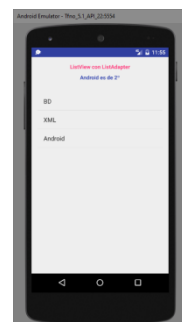
```
        textView.setText(l.getItemAtPosition(position) + " es de " + cursos[position]);
```

```
    }
```

```
}
```

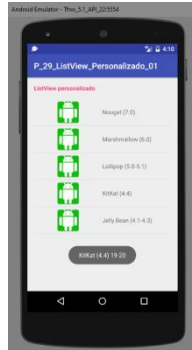
La clase "ListActivity" nos proporciona los métodos útiles para trabajar con ListAdapter.

5. Comprueba que la ejecución de la aplicación es correcta aunque se ha perdido el título de la aplicación en la "toolbar". Más adelante veremos la mejor manera de trabajar usando Fragmentos.



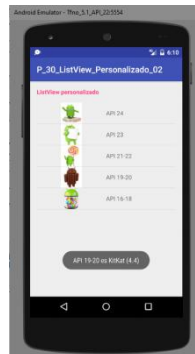
### Ejercicio 4: Proyecto P\_29\_ListView\_Personalizado\_01

1. Crea un nuevo proyecto con listView personalizado mediante layout propio. Al escoger un nombre de versión debe visualizarse mediante Toast la api correspondiente. (Recuerda drawable por tamaño usando New>Image Asset>ClipArt)

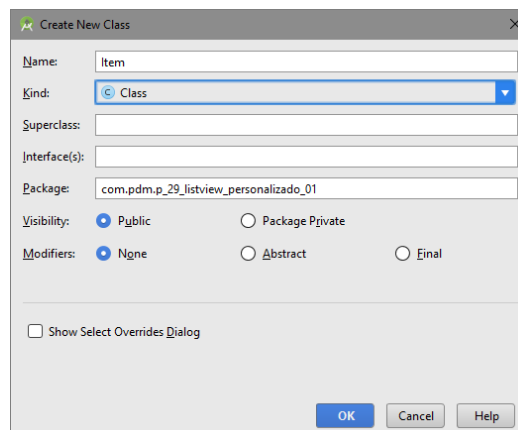


### Ejercicio 5: Proyecto P\_30\_ListView\_Personalizado\_02

1. Crea un nuevo proyecto con listView personalizado mediante adapter propio.



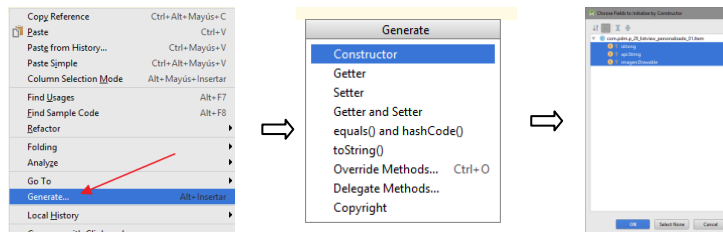
2. Los pasos son:
  - a. Diseñar el layout propio del item de la lista
  - b. Si el item de la lista contiene objetos diversos (imágenes, textos,...), para trabajar mejor, nos **crearemos una clase** que lo defina: el camino más corto, con el paquete seleccionado, botón derecho, New, Java Class



Completamos código:

```
public class Item {
    /*
     * simplemente es una clase que contendrá las variables necesarias para
     * almacenar los elementos que más tarde queremos que aparezcan en el
     * ListView. En nuestro caso:
     * long id: posición del ítem dentro de la lista
     * String api: texto que aparece en el ítem
     * Drawable imagen: imagen que aparece en el ítem
     */
    long id;
    String api;
    Drawable imagen;
}
```

Para el resto de código haremos uso de los asistentes de generación de código de AS, "click" derecho en el editor de código:



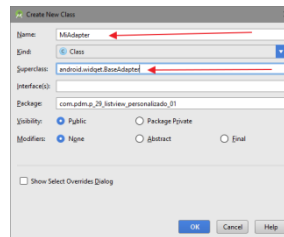
Haremos lo mismo para generar los "getter's". El código queda:

```
public class Item {
    /*
     * simplemente es una clase que contendrá las variables necesarias para
     * almacenar los elementos que más tarde queremos que aparezcan en el
     * ListView. En nuestro caso:
     * long id: posición del ítem dentro de la lista
     * String api: texto que aparece en el ítem
     * Drawable imagen: imagen que aparece en el ítem
     */
    long id;
    String api;
    Drawable imagen;

    public Item(long id, String api, Drawable imagen) {
        this.id = id;
        this.api = api;
        this.imagen = imagen;
    }
    public long getId() {
        return id;
    }
    public String getApi() {
        return api;
    }
    public Drawable getImagen() {
        return imagen;
    }
}
```

- c. Personalizar el Adapter usando el layout propio y la clase anterior

Creamos una clase para nuestro adapter:



Admitimos la sugerencia de implementar los métodos que faltan y completamos con el código señalado en rojo:

```
public class MiAdapter extends BaseAdapter {
    /*
     * Esta clase contiene dos atributos: de tipo Activity el primero y de tipo
     * ArrayList<ItemLista> el segundo. Ambos son pasados al constructor para
     * inicializar el adapter. El atributo Activity es necesario para poder
     * generar el layout que hemos creado anteriormente para nuestros item en el
     * ListView, El atributo ArrayList de items contiene los elementos que se
     * mostrarán.
     */

    private final Activity contexto;
    private final ArrayList<Item> lista;

    public MiAdapter(Activity contexto, ArrayList<Item> lista) {
        this.contexto = contexto;
        this.lista = lista;
    }

    @Override
    public int getCount() {
        return lista.size();
    }

    @Override
    public Object getItem(int position) {
        return lista.get(position);
    }

    @Override
    public long getItemId(int position) {
        return lista.get(position).getId();
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        /*
         * Debe "inflarse" 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).
         */
    }
}
```

```

LayoutInflater inflater = contexto.getLayoutInflater();
View view = inflater.inflate(R.layout.layout_item, null, true);

/*
 * Una vez que la vista para el item está preparada recuperamos el ítem que
 * vamos a mostrar utilizando el segundo parámetro que recibe el método getView
 * y el ArrayList que tenemos con los items. A continuación vamos
 * recuperando los componentes de la vista y rellenándolos con los datos adecuados.
 */
Item item = lista.get(position);

TextView textView = (TextView) view.findViewById(R.id.textView2);
textView.setText(item.getApi());

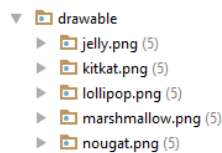
ImageView imageView = (ImageView) view.findViewById(R.id.imageView2);
imageView.setImageDrawable(item.getImagen());

// Finalmente devolvemos la vista terminada de configurar.

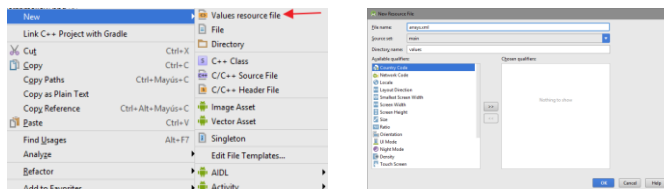
return view;
}
}

```

- d. En este ejercicio, nos interesa un [array de drawables](#) al que llamaremos logos:



Creamos un nuevo recurso tipo "values" llamado arrays.xml:



```

<?xml version="1.0" encoding="utf-8"?>
<resources>
  <array name="logos">
    <item>@drawable/nougat</item>
    <item>@drawable/marshmallow</item>
    <item>@drawable/lollipop</item>
    <item>@drawable/kitkat</item>
    <item>@drawable/jelly</item>
  </array>
</resources>

```

- e. En MainActivity que contiene el ListView asignamos el Adapter personalizado:

```

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

```
// Accedemos a los arrays de string y drawable
final Resources resources = getResources();
final String[] num_api =resources.getStringArray(R.array.num_api);
TypedArray logos = resources.obtainTypedArray(R.array.logos);
/*
 * Construimos la lista de nuestro adapter asociando cada item
 * con su número de api y su imagen
 */
ArrayList<Item> lista = new ArrayList<>();
for (int i=0;i<5;i++){
    lista.add(new Item(i+1, num_api[i], logos.getDrawable(i)));
}
logos.recycle();
MiAdapter miAdapter = new MiAdapter(this, lista);
ListView listView = (ListView) findViewById(R.id.listView);
listView.setAdapter(miAdapter);
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String[] version =resources.getStringArray(R.array.version);
        Toast.makeText(getApplicationContext(),num_api[position]+" es
"+version[position],Toast.LENGTH_LONG).show();
    }
});
}
```

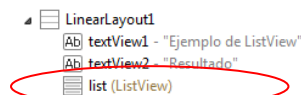
3. Recuerda drawables por tamaño usando plugin instalado.

### Nota:

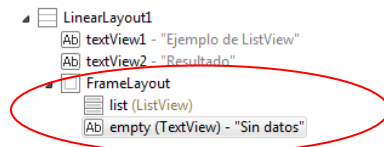
Aunque no lo hemos utilizado hasta el momento, dentro del layout contenedor de todo ListView es conveniente que exista un TextView que visualice un mensaje que avise del hecho de que la lista está vacía. En nuestros ejemplos, las listas nunca han estado sin datos (hay asignaturas) pero podría darse el caso dichos datos todavía no existan (por ejemplo, porque debe leerlos de una BD en la que todavía no hay nada, etc.).

Luego en vez de este

diseño



es mejor este otro



**Para poder utilizar el ListAdapter en el código, el TextView necesita que su id sea @android:id/empty (es decir esté ya definido por Android)**

(No es obligatorio que sea un TextView, podría ser un ImageView; lo necesario es el identificador que debe ser el predefinido por el sistema @android:id/empty para uso de lista vacía).