Guía 3: ListView y ArrayAdapter.

Objetivos

- **Aprender a utilizar ListView y su respectivo adaptador personalizado en Android.**
- **Conocer la manera correcta de implementar ListView y ArrayAdapter en Android.**
- **Implementar Listas y Adaptadores personalizados para resolver problemas en Android.**

ListView

El ListView es un grupo de Views(Vistas) que muestra una lista de elementos desplazables. Los elementos de la lista se insertan automáticamente en la lista con un Adapter(Adaptador) que toma contenido ya sea de una lista de elementos o de una consulta a base de datos, y convierte cada resultado en una vista que se dispone en la lista.

Para utilizar un ListView de manera sencilla primero lo agregamos al layout de la siguiente manera:

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

Identificamos el Tipo ListView, su respectivo id y las propiedades de ancho y alto. Únicamente con estos sencillos atributos hemos agregado el Control de la lista en el XML.

Por el lado de código tenemos lo siguiente:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //inicializando el adaptador, utilizando un prototipo por defecto
    ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1);
    //creando los datos
    String datos[] = {"dato 1","dato 2","dato 3"};
    //inicializando el control que contendra el adaptador
    ListView lista = (ListView)findViewById(R.id.lstContactos);
    //agrego los datos al adaptador -- este se encarga de mostrar los datos en el ListView
    adaptador.addAll(datos);
    //agrego el adaptador a la lista
    lista.setAdapter(adaptador);
    //TODO notifico que ha ocurrido un cambio en los los datos (agregar, modificar o eliminar)
    adaptador.notifyDataSetChanged();
}
```

Básicamente utilizamos 3 parámetros:

1. Adaptador: El cual se encarga de recibir los datos y mostrarlos, además de definir el tipo de vista que contendrá la lista.
2. Datos: Una lista o arreglo de elementos que agregamos al adaptador, cuando modificamos, agregamos o eliminamos algún elemento de este, es importante llamar a la función *notifyDataSetChanged()* para reflejar los cambios.
3. ListView: Es el control que declaramos en el XML, es el View padre que contendrá la lista de elementos, a este control se le asigna el adaptador correspondiente.

Si codificamos el ejemplo descrito obtendremos el siguiente resultado:

Básicamente es una lista de elementos de tipo String, con una plantilla predefinida para cada ítem, útil para mostrar únicamente un dato por cada View en la lista.

ArrayAdapter

Si deseamos modificar ya sea la interfaz de los ítems, la cantidad de datos, el tipo de datos, etc., en el ListView es necesario implementar un **adaptador personalizado** el cual nos brinda la libertad de personalización que necesitemos, así como un control de que se muestra en cada ítem.

Para esto es necesario haber comprendido el uso del ListView del punto anterior, ahora se agregan tres nuevos elementos:

1. Archivo XML que contendrá los controles para visualizar los datos (Ejemplo : TextView,ImageView,etc).
2. Clase personalizada para el alojamiento de los datos (Ejemplo : Persona,Contacto,etc).
3. Adaptador personalizado es una clase que hereda de ArrayAdapter que implementa el tipo de la clase personalizada (Ejemplo : ArrayAdapter<Persona>).

Podemos apreciar la aplicación de adaptador personalizado en el siguiente ejemplo:

Ejemplo [Simulación de almacenamiento de contactos]:

El código del ejemplo esta disponible en GitHub en el siguiente URL https://github.com/jonathancplusplus/Guia3_ejemplo y consiste en una app que simula almacenar contactos para esto se piden únicamente dos parámetros : el nombre del contacto y el número.

Lo primero a considerar es la creación de un tipo compuesto, dicho de otra manera una **crear clase** que nos sirva para modelar el contacto, la cual es la siguiente:

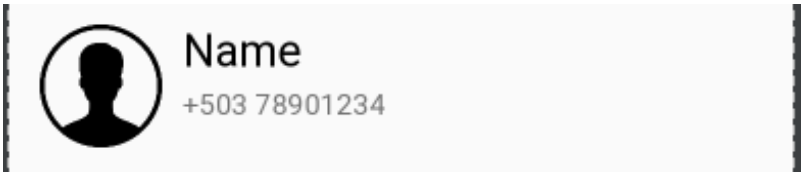
```
public class Contacto {
    public String nombre;
    public String numero;

    public Contacto(String nombre, String numero) {
        this.nombre = nombre;
        this.numero = numero;
    }
}
```

una ves finalizado procedemos a crear el siguiente archivo XML en la carpeta *app/res/layout* bajo el nombre de ítem_contacto.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="5dp">

    <ImageView
        android:layout_margin="5dp"
        android:id="@+id/imgProfile"
        android:src="@drawable/profile"
        android:layout_width="60dp"
        android:layout_height="60dp" />

    <TextView
        android:layout_marginTop="5dp"
        android:layout_marginLeft="5dp"
```

```
        android:textColor="#000000"
        android:textSize="22sp"
        android:text="Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/lblNombre"
        android:layout_alignParentTop="true"
        android:layout_toEndOf="@+id/imgProfile" />

    <TextView
        android:layout_marginTop="5dp"
        android:layout_marginLeft="5dp"
        android:text="+503 78901234"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/lblNombre"
        android:layout_toEndOf="@+id/imgProfile"
        android:id="@+id/lblNumero" />

</RelativeLayout>
```

y visualizamos el siguiente resultado:



NOTA: para obtener el mismo resultado descrito es necesario tener las imágenes y materiales que se utilizaron en el ejemplo, los cuales están incluidos en el repositorio en GitHub por lo que

se enfocara únicamente en los pasos para implementar Adaptadores personalizados.

No es necesario copiar nada de código únicamente visualizar el procedimiento que se realizo.

Una vez completado el diseño del ítem_contacto.xml procedemos a crear un Adaptador personalizado el cual hereda de ArrayAdapter:

```
public class AdaptadorContacto extends ArrayAdapter<Contacto> {

    //TODO constructor con 2 parametros : el contexto y la lista de objetos (Contacto)
    public AdaptadorContacto(Context context, List<Contacto> objects) {
        super(context, 0, objects);
    }
    @NonNull
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Obteniendo el dato
        Contacto contacto = getItem(position);
        //TODO inicializando el layout correspondiente al item (Contacto)
        if (convertView == null) {
            convertView = LayoutInflater.from(getContext()).inflate(R.layout.item_contacto, parent, false);
        }
        TextView lblNombre = (TextView) convertView.findViewById(R.id.lblNombre);
        TextView lblNumero = (TextView) convertView.findViewById(R.id.lblNumero);
        // mostrar los datos
        lblNombre.setText(contacto.nombre);
        lblNumero.setText(contacto.numero);
        // Return la convertView ya con los datos
```

Examinamos el código:

1. Primero: se implementa herencia de la clase ArrayAdapter<Clase Aquí> la cual es una plantilla que se puede sobre escribir para que sea un Adaptador del Tipo – **Contacto**.
2. Segundo: se crea un constructor que reciba el parámetro del contexto así como el conjunto de datos o lista de elementos que se van a visualizar.
3. Tercero: crear una instancia siempre del mismo tipo que definimos entre <> la cual contendrá los valores de la lista en una posición determinada.
4. Cuarto: inflar el layout que creamos para visualizar los datos.

Con estos 3 elementos definidos nos resta únicamente preparar todo en nuestra función onCreate de la clase MainActivity:

```
private AdaptadorContacto adaptadorContacto;
private FloatingActionButton btnAgregar;
private ArrayList<Contacto> contactosArrayList;


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


    btnAgregar  = (FloatingActionButton) findViewById(R.id.btnAgregar);

    contactosArrayList = new ArrayList<>();
    //Inicializando el adaptador
    adaptadorContacto = new AdaptadorContacto(this,  contactosArrayList);
    //Inicializando el listView
    ListView listView = (ListView) findViewById(R.id.lstContactos);
    //seteando el adaptador al listview
    listView.setAdapter(adaptadorContacto);
    btnAgregar.setOnClickListener((v) → {
            //preparo el activity
            Intent intent = new Intent(MainActivity.this,ContactoActivity.class);
            //inicio el activitiy y quiero como mensaje un RESULT_OK
            startActivityForResult(intent,GUARDADO);
    });

    //cuando de click sobre un item
    listView.setOnItemClickListener((parent, view, position, id) → {
            //TODO cuando toco un contacto muestro un mensaje
            Toast.makeText(MainActivity.this,"Contacto "+contactosArrayList.get(position).nombre,Toast.LENGTH_SHORT).show();
    });
}
```

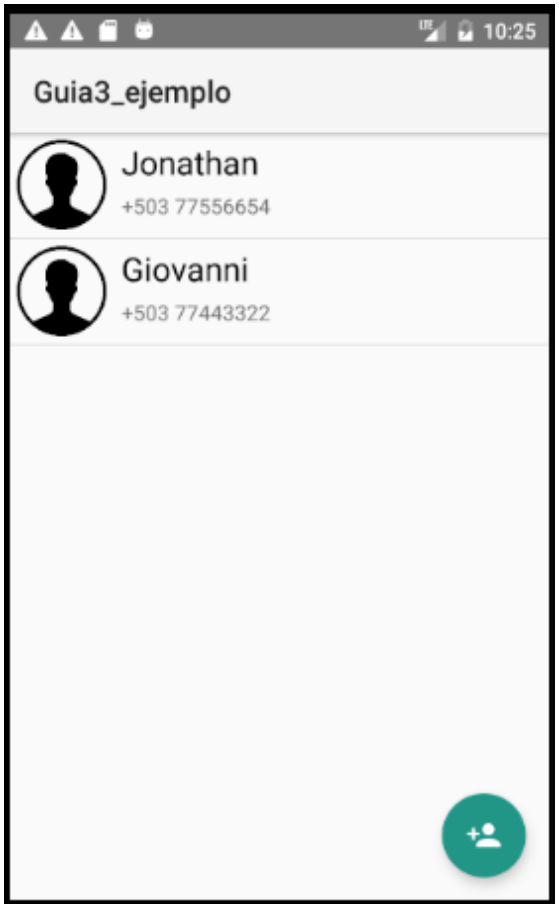
Se aprecia que ahora se envía una lista de elementos del tipo **Contacto** en el **adaptadorContacto**.

En este ejemplo se preparo además un botón del tipo FloatingActionButton que nos sirve para llamar a otra activity que se encarga de pedir los datos del **Contacto**, por lo cual se hace uso de *startActivityForResult* y se espera el resultado.

Nota: Para manejar los eventos onClick sobre cada uno de los ítems se utiliza para función setOnClickListener del control ListView, y se comporta similar al control Button en su evento onClick.

La aplicación ya finalizada se aprecia de la siguiente manera:

Visualizar la lista de elementos



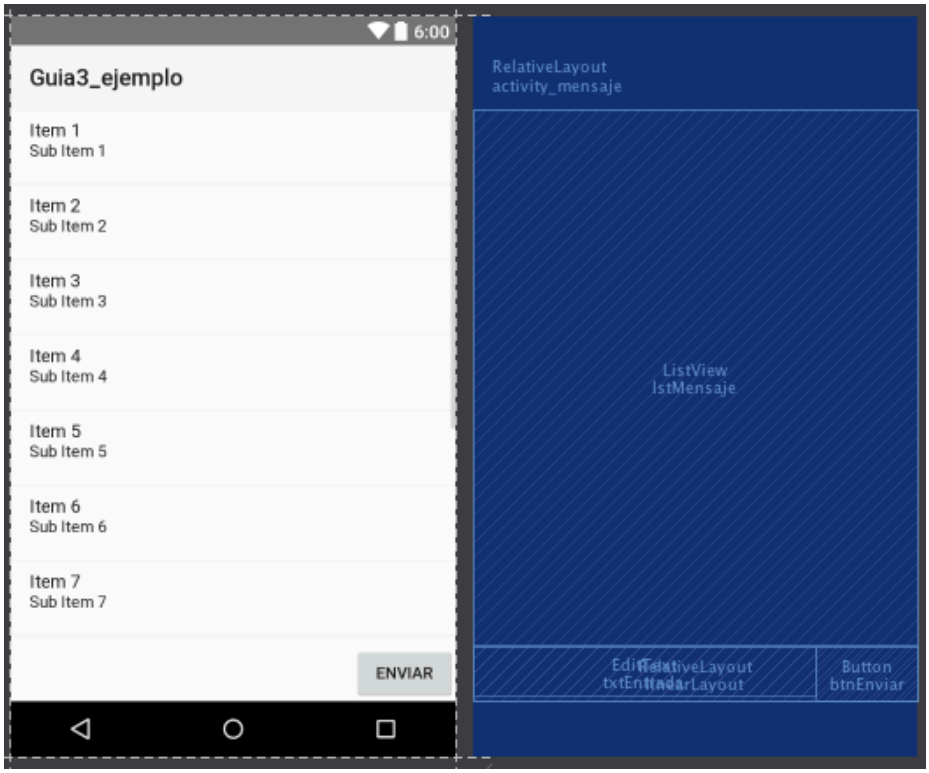
Al pulsar el botón lanza otra Activity

Como resultado obtenemos una aplicación que guarda una lista de objeto personalizado (Contacto) y lo muestra en un ListView mediante el layout y el uso de adaptadores personalizados.

Ejercicio:

Modificar el código proporcionado en GitHub sobre el ejemplo agregando las siguientes características:

1. Otra Activity que se llamara cuando se pulse sobre un ítem de la lista Contacto ponerle de nombre *MensajeActivity*.
2. En MensajeActivtity agregar un ListView además de un EditText y un Button para simular enviar MSJ.



3 crear una clase llamada Mensaje que contendrá los siguientes parámetros además del constructor:
   - String fecha
   - String contenido
4 Crear un adaptador personalizado para Mensaje y codificarlo.
5 En Mensaje Activity programar la funcionalidad de los controles, el botón enviar únicamente agregara a la lista de Mensaje un nuevo elemento que sea la fecha actual y el contenido del EditText, y actualizara la lista utilizando la función *notifyDataSetChanged()* perteneciente al adaptador Mensaje.

Nota: para que el estudiante comprenda el ejercicio se dispone del APK completo, para que visualice como opera, esto con el fin de evitar dudas, este APK será invalido si se entrega como tarea.

https://www.dropbox.com/s/yiz693rc8vn9ez6/APP Guia3.apk?dl=0

Resultado:


11:11
Guia3_ejemplo
Fri, 1 Sep 2017, 23:10
este
Fri, 1 Sep 2017, 23:10
es
Fri, 1 Sep 2017, 23:10
un mensaje
Fri, 1 Sep 2017, 23:11
de prueba
ENVIAR