

Desarrollo de aplicaciones con Android

Índice

Semana 1.....	2
Aplicando el tema Material Design.....	2
Ejercicio: Integrando Material Design a tus proyectos.....	5
Semana 2.....	10
¿Qué son las Activities?.....	10
Ejercicio: creando una actividad.....	11
Declarando Views (Elementos de interfaz de usuario) por XML.....	17
Layouts y Raised Button.....	18
Floating Action Button.....	32
Snackbar.....	39
Refresh Indicator.....	47
CardView.....	54
Tipos de layouts.....	60
Qué son los layouts.....	60
Layouts en android.....	60
Lineal layout.....	61
Relative Layout.....	62
List View.....	63
Grid View.....	64
Frame Layout.....	66
Ciclo de vida de un activity.....	66
Ciclo de vida de un activity.....	66
Métodos Callbacks.....	68
Intents.....	71
Iniciando una actividad. Intents.....	71
Tipos de intents: explícitos e implícitos.....	71
Ejemplo de intents explícitos e implícitos.....	74
Terminando Activities.....	89
Semana 3.....	91
RecyclerView.....	91
Ejemplo de RecyclerView.....	94
RecyclerView onClick.....	101
Botón de Like en RecyclerView.....	103
Añadiendo un App Bar Material Design y Ejemplo.....	106
Navegación hacia atrás.....	114
para qué y cómo.....	114
Ejemplo: Navegación hacia atrás.....	116
Semana 4.....	120
Menú de opciones.....	120
Ejemplo de menú de opciones.....	121
Menú de contexto.....	129

Ejemplo: Menú de contexto.....	129
Menú Pop-up.....	133
Ejemplo: Menú Popup.....	133
Integrando Action Views.....	139
Fragments.....	140
¿Por qué usar Fragments?	140
Creación de Fragments.....	143
ViewPager y Fragment.....	148
Añadiendo RecyclerView en un Fragment.....	159
Añadiendo un Fragment desde un layout.....	162

Semana 1

Aplicando el tema Material Design

Aquí vamos a aprender cómo integrar todos los elementos de material design, vamos a ver cómo se compone un activity, todo lo que necesitas saber sobre ella, los fragments, vamos a ver seguridad en Android para que todas tus aplicaciones estén muy bien diseñadas.

Además, vamos a ver también cómo manejar todo el almacenamiento interno, externo, bases de datos, etcétera, tenemos muchísimas cosas por ver, así que pues, ¡comencemos!

Ha llegado la hora de aplicar material design a nuestros proyectos.

Lo primero que debes saber es que material design fue introducido en Android 5.0 en el API 21.

Esto quiere decir que a partir de esta versión se han involucrado nuevos widgets, transiciones, animaciones, todas esas cosas las tendremos disponibles ya para nuestros proyectos.

Material design nos va a ayudar a poder personalizar nuestra aplicación tanto como la identidad de nuestra marca, así que es sumamente flexible.

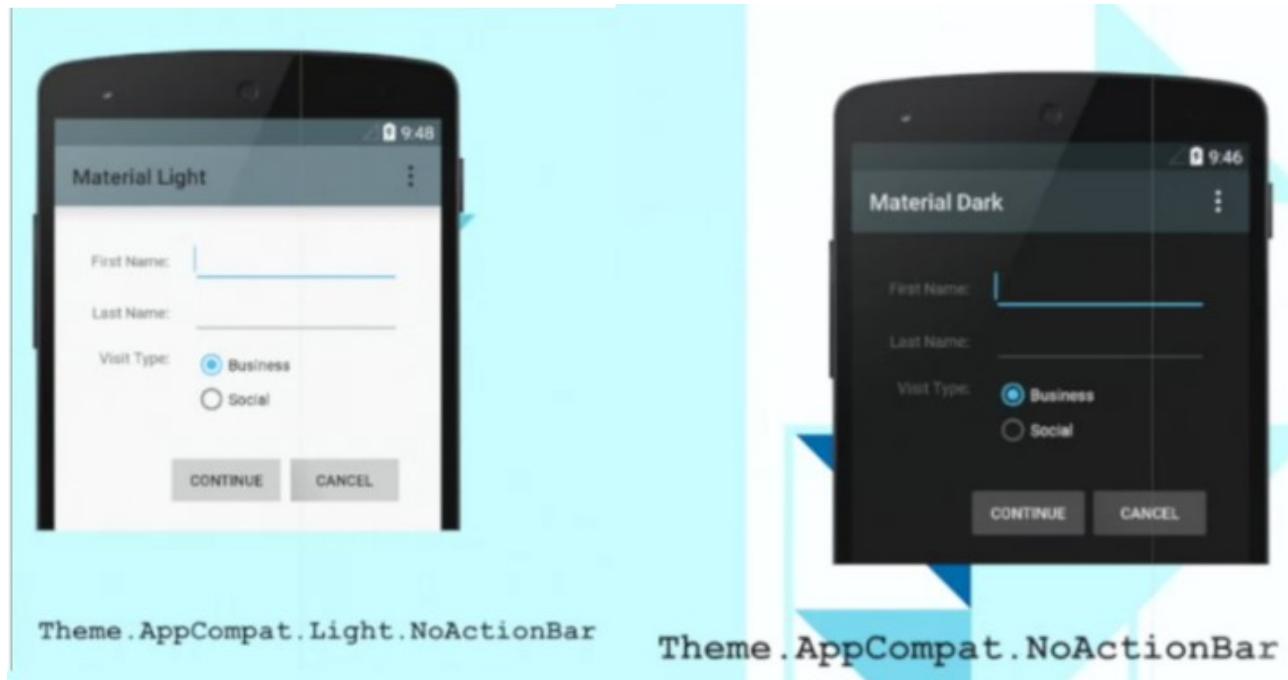
Esto se logra gracias a la introducción de dos temas.

El primero un tema claro y el segundo un tema oscuro, esto dependiendo de cómo quieras personalizar tu aplicación.

Probablemente el tema de tu aplicación puede ser algo que los elementos requieran ser todos oscuros, o de lo contrario que sea todo claro.

El primero que define un tema claro, como lo está haciendo en esta imagen, es el tema theme.appcompat.light.nodarkactionbar.

Y el segundo tema que nos va a definir un color oscuro en nuestra aplicación es theme.appcompat.noactionbar.



Estos temas se pueden configurar en nuestro archivo styles.xml.

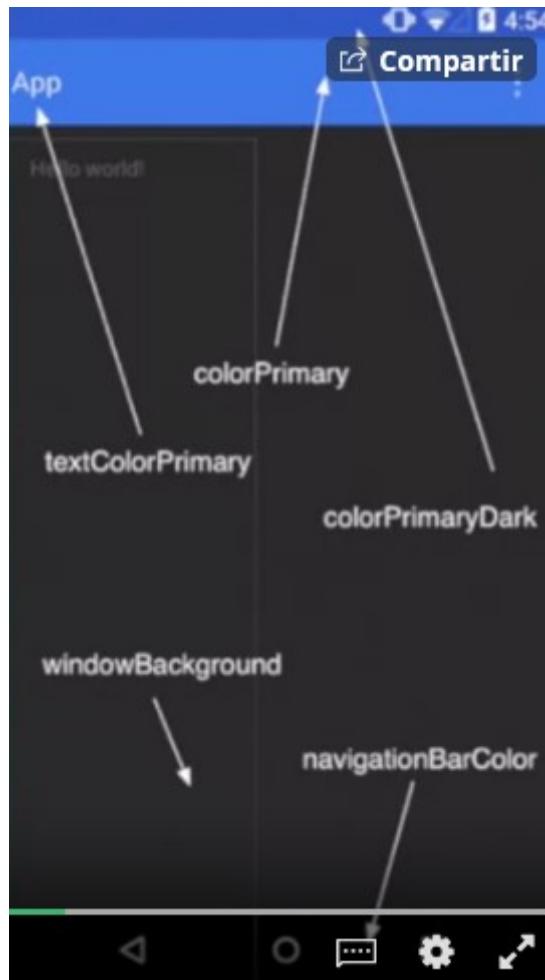
```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Como puedes observar, tenemos nuestra etiqueta styles y ahí es donde indicamos el tema que queremos aplicar para nuestra aplicación.

Además dentro de nuestra etiqueta styles, ahí podemos definir y podemos alterar o configurar personalizadamente algunos atributos que queremos adicionalmente que tenga nuestra aplicación.

Como los colores principales que vimos en nuestro curso anterior, nuestro color primario, nuestro secundario, nuestro primario oscuro, etcétera, además de otros que como puedes observar por acá.

En esta imagen, se visualizan más o menos algunos de los colores que podemos estar alterando, que podemos estar configurando personalizadamente para nuestra aplicación.



Podemos configurar la barra de navegación que está en la parte superior, podemos configurar también el texto que se muestra ahí, además podemos configurar también el color de fondo que tenga nuestra aplicación, aunque eso también va a depender del tema que escojas, oscuro o claro.

Y además también podemos configurar la parte que está abajo que es nuestro navigation bar, para darle un estilo más cool a nuestras aplicaciones. Todo esto lo hacemos en el archivo styles.xml definiendo el tema que utilizarás con material design.

Algunas de las cosas que hemos hablado hace un momento, solamente están disponibles cuando configures tu proyecto desde una versión 5.0 de Android.

Como sabemos toda esta filosofía de material design, fue introducida a partir de Android Lollipop, por lo tanto muchas características no van a estar disponibles en versiones anteriores a Lollipop.

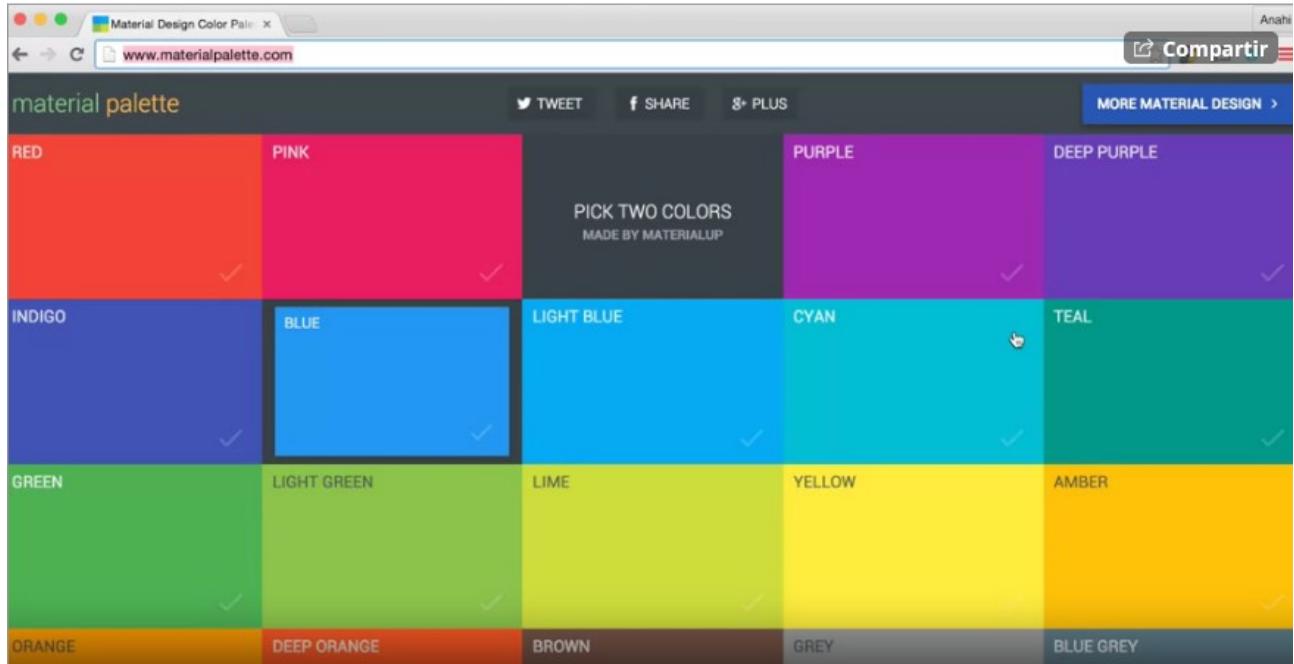
No te preocunes, si nosotros todo el tiempo estamos utilizando como tema principal de nuestra aplicación theme.appcompat, podemos gozar de la mayoría de estas.

Vamos a ver cómo se hace con un ejemplo.

Ejercicio: Integrando Material Design a tus proyectos

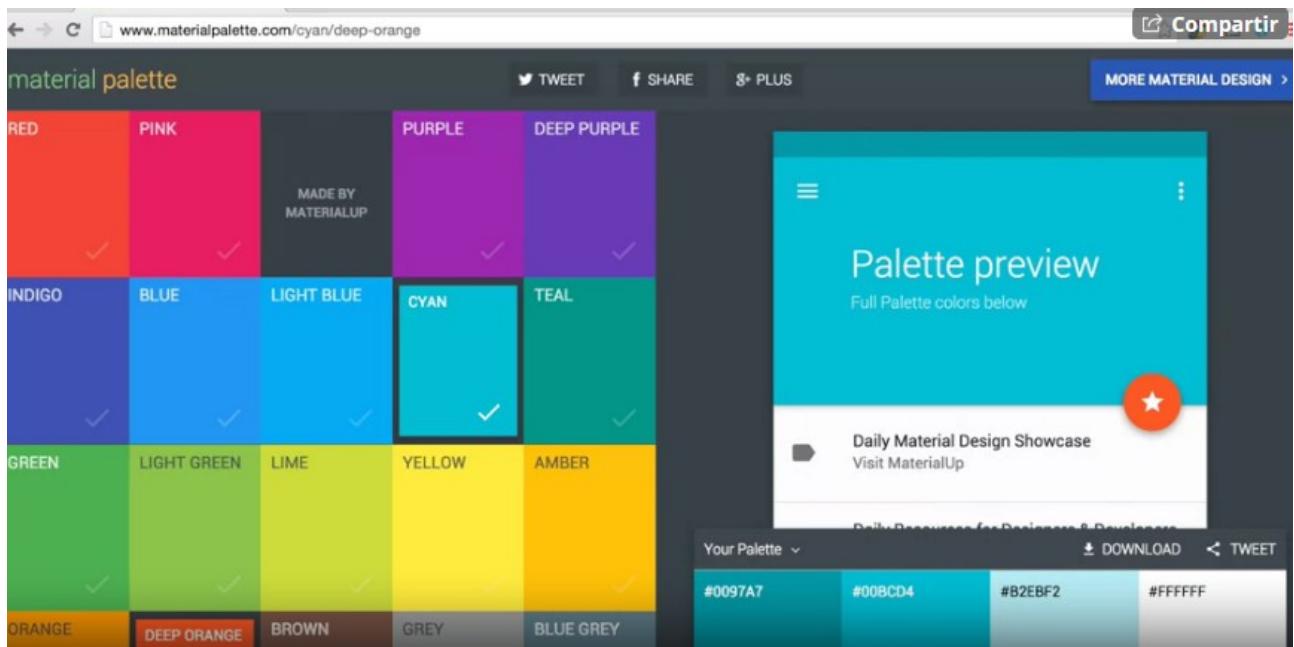
Bien, para integrar material design en nuestros proyectos, lo primero que tenemos que definir es la composición de colores que va a tener.

Para esto tenemos un recurso en internet que nos va a hacer la vida super sencilla, este recurso se llama materialpalette.com.



Cuando entremos a materialpalette.com, lo primero que se va a mostrar es todo un grid de colores, todo un grid de colores donde nos va a permitir pues seleccionar dos colores específicos.

Yo puedo seleccionar por ejemplo para mi aplicación quiero este y quiero que también se vea este.



El primer color que seleccionamos es el color primario, o es el color principal que va a tener nuestra aplicación, y el segundo color que seleccionemos será nuestro color de acentuación.

Entonces como observas aquí en la pantalla, pues esto ya nos está haciendo todo, absolutamente todo el trabajo puesto que ya nos está generando a partir del color primario, nos está generando también un dark primary color, que es este color que va a tener aquí en la parte superior, en la barra, nuestro accent color ya lo está tomando también, para todos nuestros botones, algunos widgets, elementos que tengan que ser acentuados. Y además, no solo eso, sino también nos está generando todos los textos y todos los elementos que puede contener una aplicación.

Esto es muy importante porque en las métricas de diseño de material design en la página oficial, en la documentación oficial, es un poco complicado a partir de cierto color tomar tantos grados de opacidad para definir el color de un texto o para definir otro elemento, etc.

Entonces todo esto ya es resuelto con este sitio en internet materialpalette.com, y entonces ya no debemos tener problemas por ello.

Bien, ya tengo cómo va a lucir mi aplicación, yo puedo dejarla así o puedo seguir jugando con otro color, puedo decirle que sea el púrpura y que sea el tono naranja, o a lo mejor quiero un rosa, con azul o un color lima, etc.

Yo voy a seguir escogiendo este y este que está aquí, yo quiero que así se vea mi aplicación.

Bien. Entonces una vez que ya tengo definido esto, puedo ir a mi Android Studio y voy a darle start a new android studio project, voy a seleccionar nuestra aplicación va a tener un nombre curioso, le voy a colocar Petagram, más adelante les voy a explicar por qué se va a llamar así.

Bien. Lo que tenemos que definir ahora como mínimo SDK es la versión mínima que queremos que tenga, que sea compatible nuestra aplicación. Yo voy a colocar, puedo colocar el API 14 o el API 15, que es básicamente la versión de Ice Cream Sandwich, y vamos a darle siguiente.

Y ahora vamos a colocar empty activity. Le damos siguiente, colocamos el nombre de nuestro activity y finish. Vamos a esperar a que Gradle comience a construir todo el proyecto, a que comience a unir todas las dependencias y que finalmente ya nuestro proyecto quede construido.

En la parte inferior puedes observar cómo es que Gradle está construyendo el proyecto y puedes ver también cuando ya haya terminado.

No te espantes cuando de primera instancia veas estos elementos en rojo, simplemente hay que esperar a que el proyecto termine de construirse.

Bien. El proyecto ya está terminado, en la parte inferior ya no veo nada, y ahora como dijimos voy a ir a mi archivo styles Mi archivo styles lo vas a encontrar en la carpeta res, en values y aquí ya está.

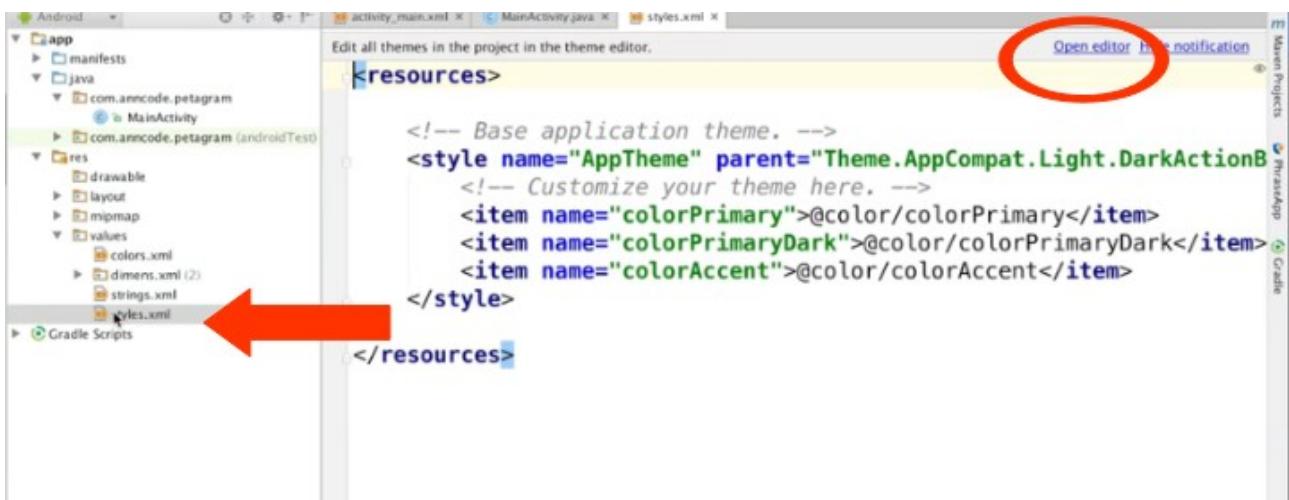
Archivo styles:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

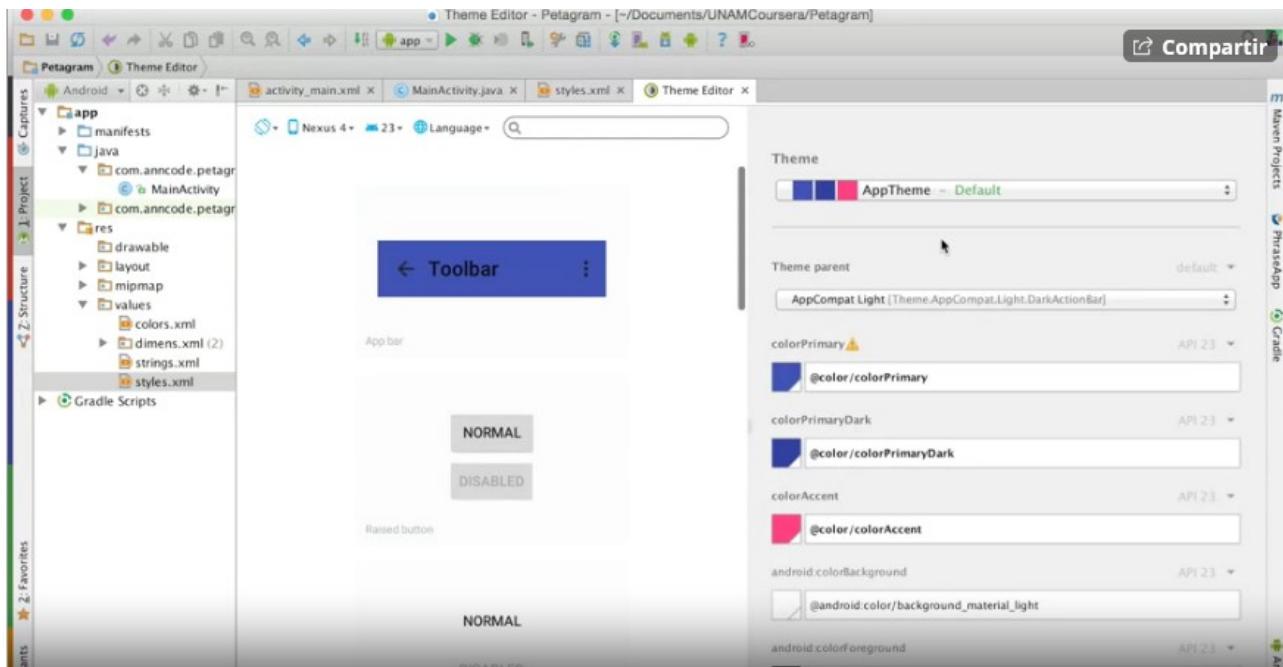
Como observas, automáticamente ya estamos heredando del tema Theme.AppCompat y el tema que ahorita está definido es un tema light, un tema light que sí tiene un DarkActionBar. Recuerdas que en los anteriores colocábamos que decía NoActionBar.

En secciones consecutivas vamos a ver por qué vamos a colocar NoActionBar.

Bien, si observas por aquí tengo una notificación, como una notificación que me dice open editor, ¿no?



Yo puedo abrir el editor y lo que estoy observando es un preview de todos mis elementos que puedo tener, dependiendo de la combinación de colores que en este momento, por default ya está.



Aquí se observan con parámetros, con variables, de una forma más legible, más sencilla, color primary, color primary dark, color accent, acá están todos los atributos que yo puedo editar, puedo estar aquí modificando, colorBackground, foreground, material light, un color black para el navigationBarColor, statusBarColor, textColorPrimary, etcétera, todo lo que colocamos nosotros por aquí, todo lo que está definido aquí en estos elementos.

Tengo un light primary color que va a corresponder al texto que está por aquí, tengo un texto que va a corresponder al texto que está por acá.

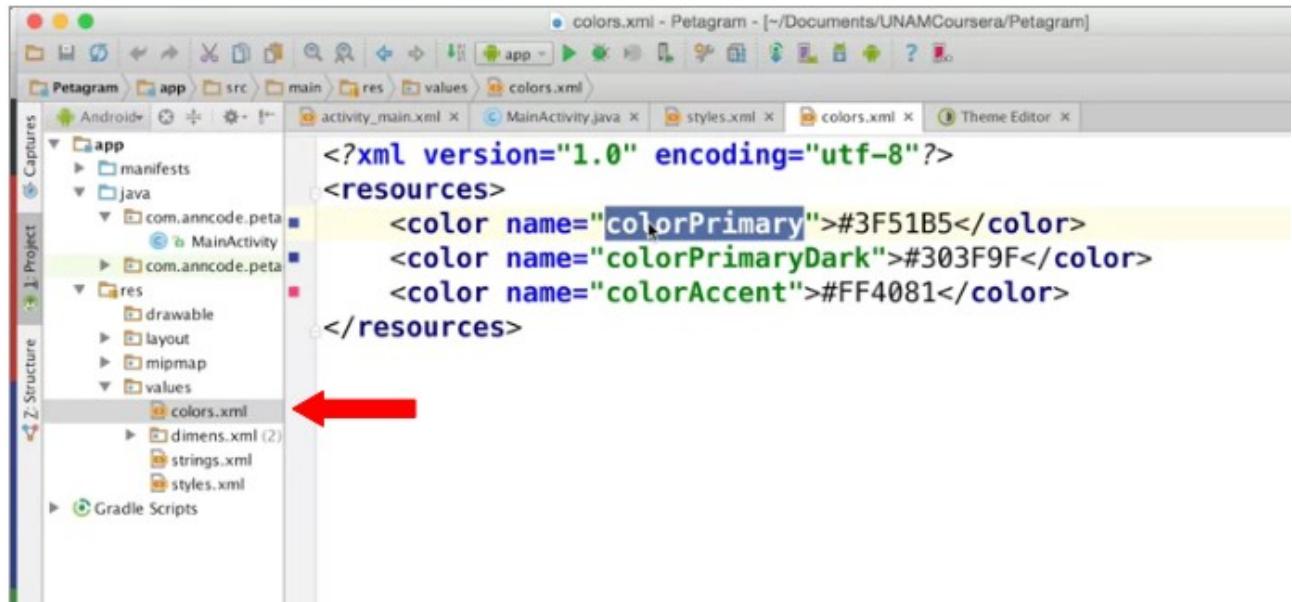
Y todos estos elementos los podemos definir en nuestro archivo styles a partir de su parámetro correspondiente.

Entonces, si voy a mi layout, mi layout debe verse así, dice Petagram, y esta, tiene una pantalla en blanco, no tiene ningún elemento. Bien, vamos a configurar esto para que quede de acuerdo a la identidad de nuestra marca.

Bien, ya tenemos aquí 3 items, que son colorPrimary, colorPrimaryDark y colorAccent.

Estos items están respondiendo a un recurso que está ubicado en la carpeta color, color y en el archivo colors, color que es un recurso y entonces contiene por aquí una variable que se llama color primary.

Vamos a abrir nuestro archivo colors y ahí efectivamente vemos nuestra variable colorPrimary, acá tenemos un colorPrimaryDark y un colorAccent.



Todos estos colores están definidos a partir de su código hexadecimal, entonces y de este lado que tenemos aquí podemos ver un preview del color y si yo quiero pues puedo aquí mismo modificarlo.

Entonces, si yo ya tenía por acá mis recursos, simplemente selecciono el, por ejemplo primary color y al seleccionarlo ya nos muestra que esto ya se copió en el portapapeles. Entonces, el color primario y ya puedo utilizarlo, simplemente voy a dar un control v o un command v y, como ves por acá, ya está el color seleccionado.

Ahora, para un color primary dark, vamos a hacer lo mismo, tenemos por aquí nuestro primary dark, ahí está, ya está copiado y podemos colocarlo aquí.

Y para el color de acentuación también haremos lo mismo, colocamos esto y aquí está.

Perfecto, hasta ahora podríamos nosotros seguir configurando nuestro tema tanto como querramos, pero hasta ahorita solamente vamos a definir estos 3 colores.

Al mismo tiempo, en este archivo de colores, podemos definir por ejemplo para un texto para el título.

Un textoTitulo y puedo definir otro color que sea un color blanco por ejemplo para el título y aquí puedes seguir definiendo tantos colores quieras.

Ahora, en nuestro archivo styles, como observas también esto ya fue aplicado.

Aquí se está haciendo referencia a este recurso colorPrimary y aquí se ve también ya el color asignado.

Para este parámetro colorPrimaryDark que corresponde a este tema, también ya está asignado colorPrimaryDark, y para el color de acentuación, también está designado el color de acentuación.

Ahora si vemos nuestro layout cómo va, cómo se va viendo, bueno, vemos que por lo menos los colores ya cambiaron, ya no tenemos el azul que teníamos antes.

Ahora tenemos este color azul más clarito que le da una vista más fresca a nuestra aplicación.

Y entonces, esto es lo primero que tenemos que hacer para empezar a integrar material design a nuestras aplicaciones, dejar el tema que necesitamos, hasta el momento vamos a dejar este tema con DarkActionBar, escogimos un tema light y configurar, por lo menos, nuestros 3 colores principales de nuestro tema.

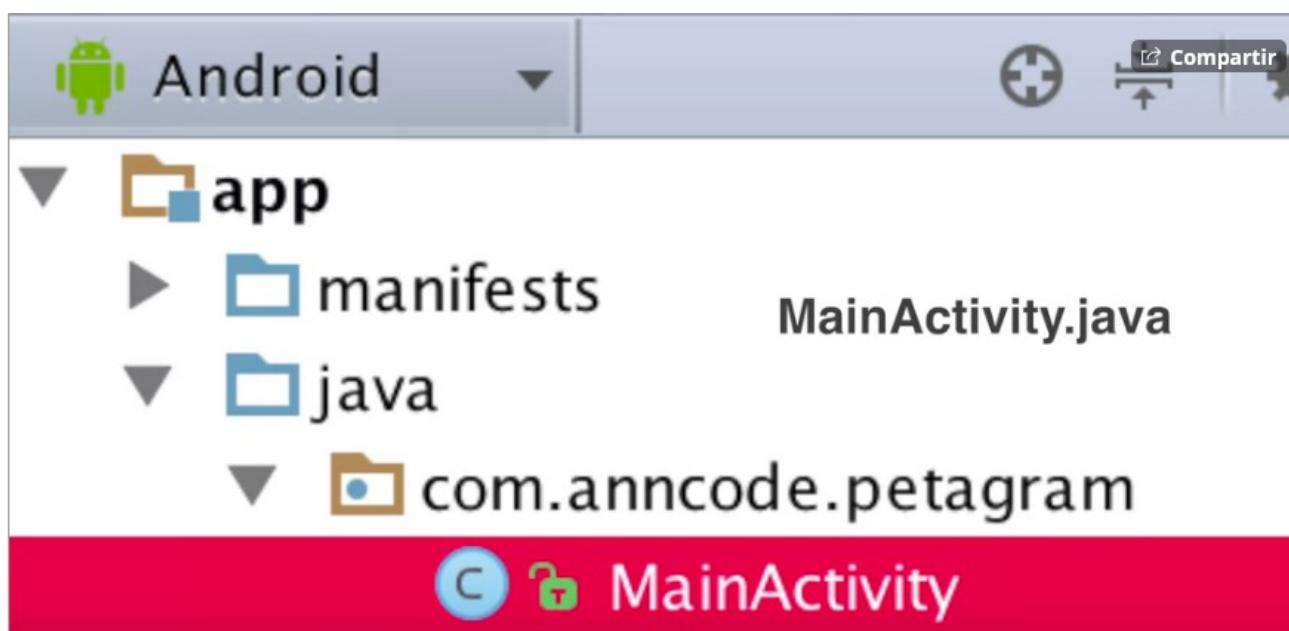
De esta forma es como hemos terminado de integrar material design en nuestro proyecto y ahora vamos a nuestro siguiente módulo para darle más vida a nuestra aplicación y comenzar a crear actividades.

Semana 2

¿Qué son las Activities?

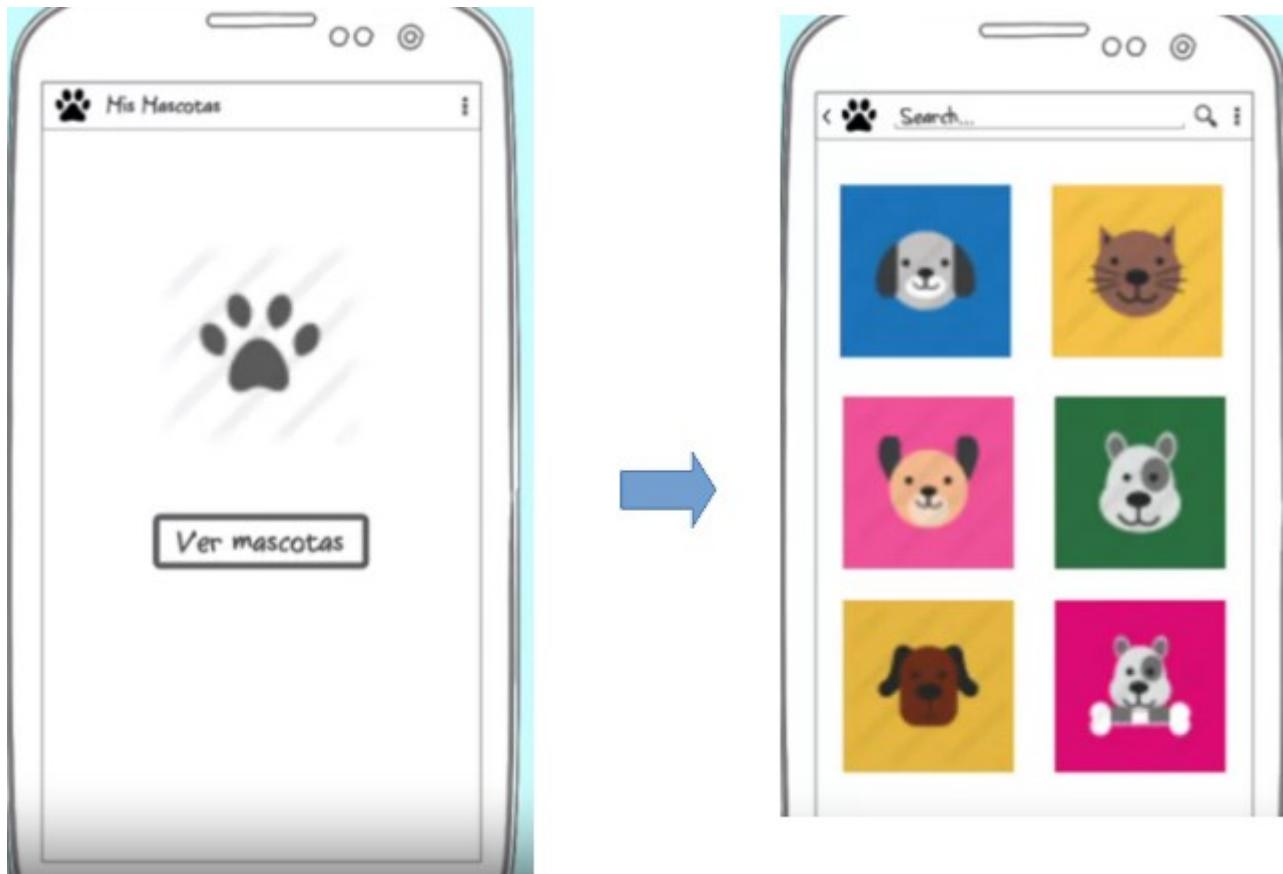
Los activities son componentes de una aplicación que provee una interfaz para que los usuarios interactúen con ellas. O sea, dibuja la interfaz de usuario.

Por lo general, una aplicación se compone de muchas pantallas o de muchas actividades, donde se debe nombrar una sola como la actividad principal (MainActivity)



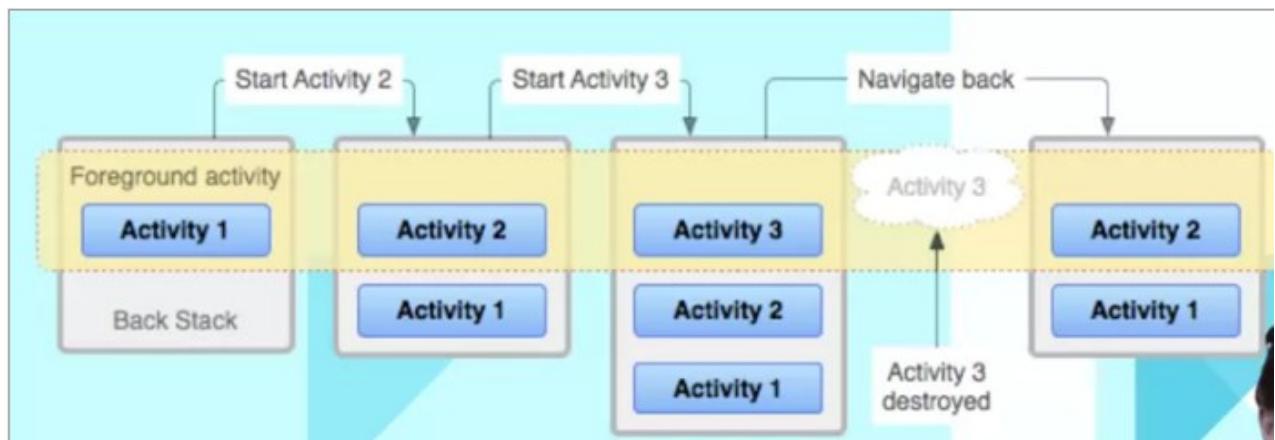
Ésto significa que es la actividad que se ejecute cuando la aplicación se inicie.

Por otra parte, una actividad puede iniciar otra actividad cuantas veces sea necesario, por ejemplo, en una actividad, al presionar un botón inicie otra actividad.



Cada vez que se inicia una nueva actividad, éstas se van acumulando en una pila donde las actividades anteriores quedan detenidas esperando a ser reanudadas.

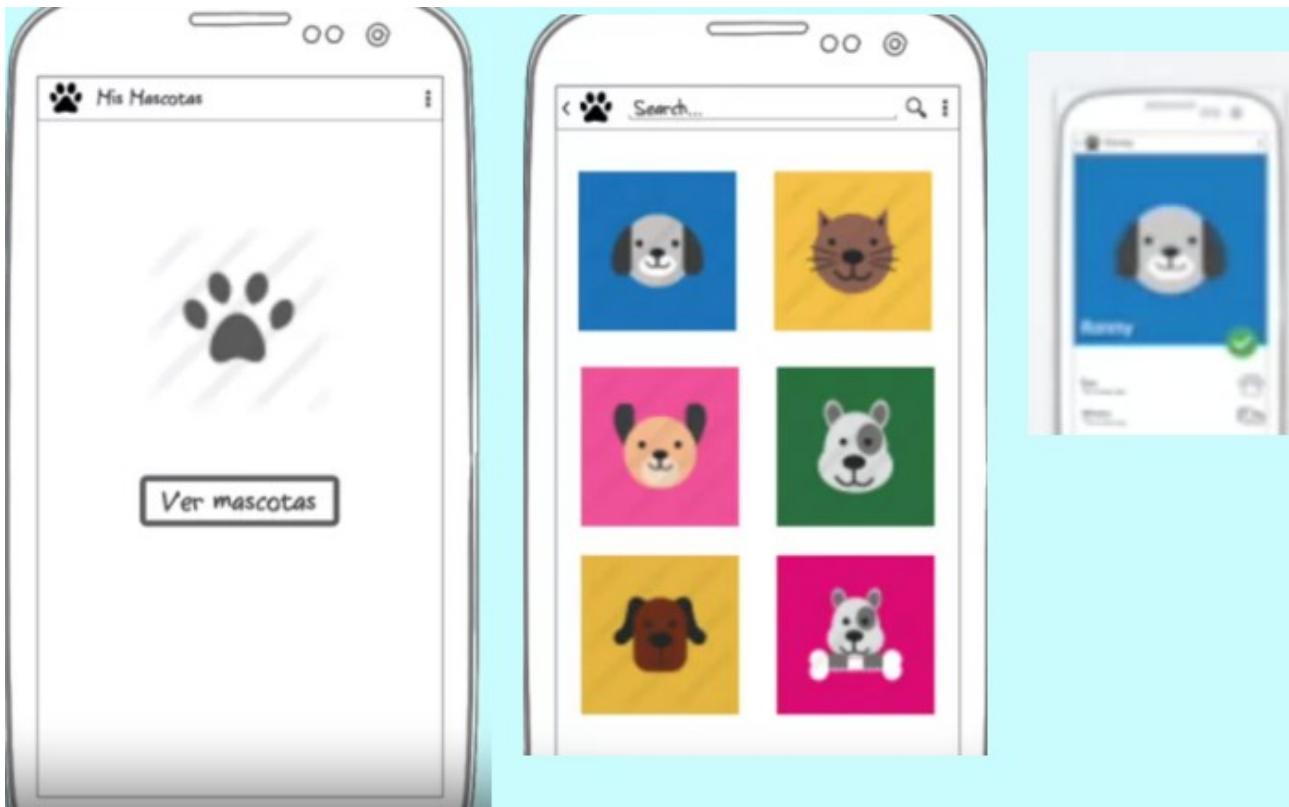
Al estar en una pila, seguirá la filosofía de que la primera actividad al entrar, va a ser la última en salir.



Por ejemplo, si se tienen 3 actividades en la pila, y se presiona el botón de back en el teléfono, va a ir a la actividad 2 (pasa a foreground) y la actividad 3 es destruida.

Ejercicio: creando una actividad

Tengo ahora en pantalla la maqueta que vamos a estar utilizando para esta aplicación. Y como observamos, pues tenemos tres activities, tres pantallas.



Tres pantallas que hasta el momento para nuestra aplicación, la van a componer. Entonces tengo como una pantalla de inicio, una pantalla donde se listarán todas las mascotas que tengo. Y una pantalla donde se muestran las fotos o donde se muestra el detalle de esa mascota que he elegido.

Y donde nosotros creamos nuestro proyecto por primera vez, por default el ID ya nos crea una primera actividad.

Una primera actividad se va a componer de una clase de Java que se llama main activity. Y también se va a componer de un layout, un layout que vas a encontrar en la carpeta res, layout y que se llama activitymain.xml.

Bien activitymain.xml y nuestra clase de Java. Estos en conjunto forman una actividad.

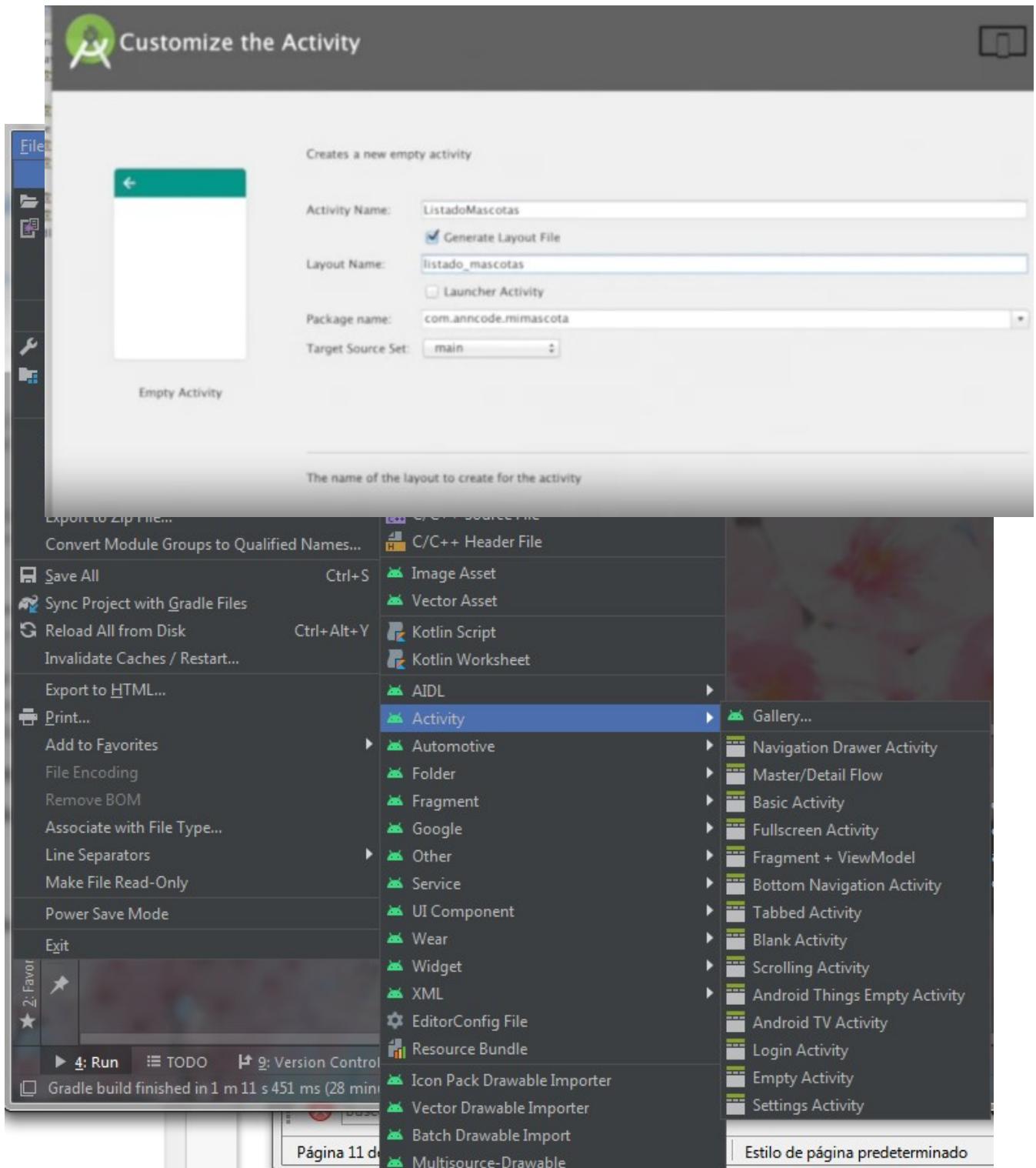
Entonces digamos que yo ya tengo la primera pantalla, lo primero que estoy viendo aquí, aunque todavía no he definido mi interfaz como se muestra.

Entonces simplemente voy a mostrarte como crear estos dos activities más.

Voy a colocarme aquí, puedo estar en cualquier nivel del proyecto, o puedo ir simplemente al menú file, donde dice new. File → new → Activity.

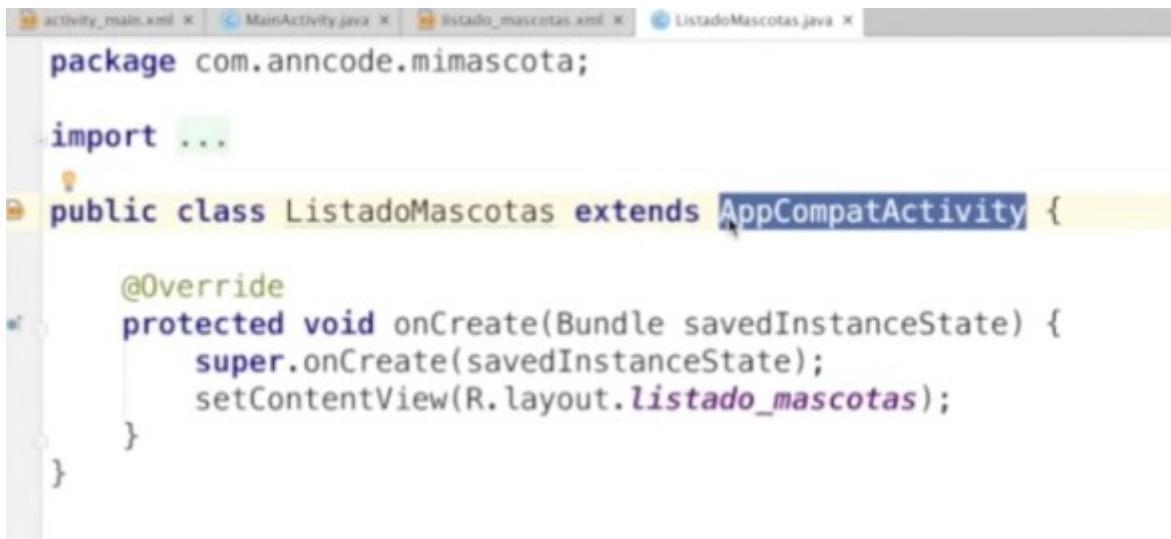
Voy a darle empty activity, y lo que nos está pidiendo ahora es el nombre de la actividad.

Recuerda que la actividad la compone mi archivo de Java y también de mi layout.



Entonces voy a colocar aquí Listado de mascotas. Entonces así es cómo se va a llamar mi archivo de Java y mi layout.

Entonces aquí yo puedo decidir en qué package quiero colocarlos, y le voy a dar Finish. En este momento, empieza a trabajar el ID, y automáticamente me crea mi archivo de Java.



```
package com.anncode.mimascota;

import ...

public class ListadoMascotas extends AppCompatActivity {

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

Y también por acá en el layout encuentro mi layout correspondiente a este.

Entonces puedo tener otra forma también de crear estas actividades.

Como observas, ListadoMascotas es una clase que se ha creado como si crearas una clase normal.

Y que ésta está heredando de la clase AppCompatActivity.

AppCompatActivity es una clase que finalmente hereda de nuestra clase padre, de nuestra clase padre activity.

Entonces esta clase es como si a pesar de que una clase hija de activity, pues en automático contiene todos los métodos de una actividad.

Entonces esto es algo que vamos a estar haciendo mucho en android.

Básicamente vamos a estar utilizando clases ya existentes heredándolas en nuestras nuevas clases e implementando los métodos que contienen cada una de éstas clases.

Si recordamos los conceptos de herencia, sabemos que cuando una clase hereda de otras, automáticamente posee todos los atributos y todos los métodos de la clase padre.

Entonces aquí yo voy a tener que automáticamente ya se está sobre escribiendo un método.

Un método, que es muy importante y muy interesante, es nuestro método onCreate.

Nuestro método onCreate nos va ayudar, como su nombre lo dice, a crear nuestra actividad.

Es decir, cuando una activity llame a otra, en ese momento va suceder un evento o va suceder la creación de mi actividad. Cuando suceda eso, este método se va ejecutar automáticamente.

Y entonces lo que hace este método, es que en primer lugar tiene super, que quiere decir que está llamando a la super clase, a su clase padre.

Y está llamando al método onCreate de la super clase. Es decir, está reutilizando todo lo que tiene la clase activity para crear esta clase que ahora yo tengo.

Después tenemos esta instrucción, `setContentView`, este método, lo que nos va a ayudar es que todo lo que yo tengo en mi layout.

Todo lo que vive ahí en mi carpeta de layout, que es el código xml donde están todos mis views, mi interfaz gráfica.

Automáticamente le estoy diciendo que eso es lo que va a mostrar cuando se ejecute ese evento de crear la actividad. Entonces que comience, que sucede, se empieza a crear la actividad, automáticamente llama a su layout correspondiente y entonces muestra nuestra interfaz gráfica. Eso es lo que está sucediendo con esta línea de código.

En otras palabras, en términos más de programación, se dice que se está seteando una vista.

Entonces, yo después de esta línea de código, puedo comenzar a trabajar mis views o todos los elementos que componen mi interfaz gráfica. Si yo no tuviese esta línea de código, no puedo estar referenciando, o no podría estar yo llamando a elementos que estén dentro de esta interfaz.

Entonces de esta forma, muy sencillamente con el wizard, con el asistente de Android Studio puedo crear una actividad.

Si quisieras crearla de otra forma, bueno, puedes simplemente crear una clase. Click derecho, new, java class.

La última actividad que nos hace falta es ésta. La del detalle de mi mascota.

```
package com.anncode.mimascota;

import android.os.Bundle;
import android.os.PersistableBundle;
import android.support.v7.app.AppCompatActivity;

/**
 * Created by anahisalgado on 10/03/16.
 */
public class DetalleMascota extends AppCompatActivity {

    @Override
    public void onCreate(Bundle savedInstanceState, PersistableBundl
        super.onCreate(savedInstanceState, persistentState);

    }
}
```

Le vamos a poner `DetalleMascota`. Puedo estar heredando `extends` de `AppCompatActivity`, hasta ahí. Puedo sobre escribir mi método `onCreate`. Puedo simplemente estar escribiendo por ahí la palabra `onCreate` y automáticamente reconoce esto y lo implementa super rápido.

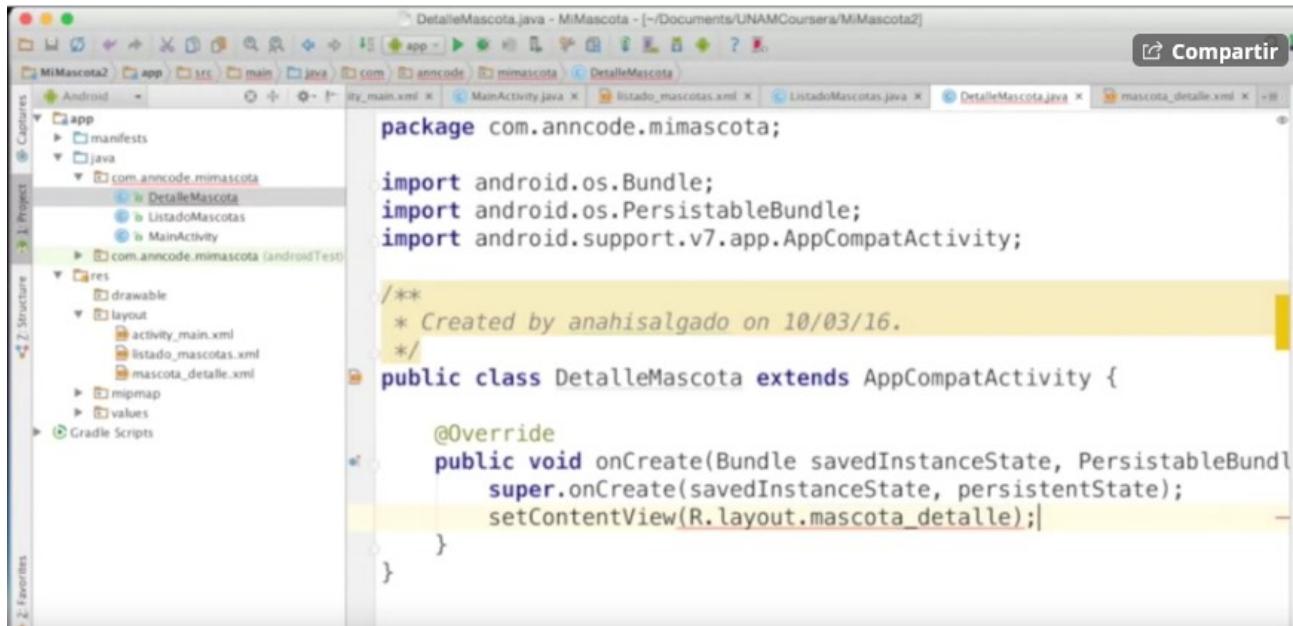
Entonces ya está aquí super, bien.

La siguiente instrucción sería setear la vista. ¿Cuál vista?, todavía no la he creado. Entonces puedo ir a mi carpeta layout, darle click derecho, new Layout resource file, y ahí colocar, se debe llamar mascota_detalle.

Recuerda que los nombre de layout son generalmente al revés. Al revés de como los definimos en nuestro en nuestro archivo de Java.

Entonces en DetalleMascota, ahí le puedo colocar la instrucción setContentView, y entonces a través de mi archivo R.

Es un layout a lo que estoy accediendo, R.Layout.mascota_detalle, y listo.



```
DetalleMascota.java - MiMascota - [~/Documents/UNAMCoursera/MiMascota2]
package com.anncode.mimascota;

import android.os.Bundle;
import android.os.PersistableBundle;
import android.support.v7.app.AppCompatActivity;

/*
 * Created by anahisalgado on 10/03/16.
 */
public class DetalleMascota extends AppCompatActivity {

    @Override
    public void onCreate(Bundle savedInstanceState, PersistableBundle persistentState) {
        super.onCreate(savedInstanceState, persistentState);
        setContentView(R.layout.mascota_detalle);
    }
}
```

Eso es lo que se está aquí logrando.

Un paso más que es muy importante, es que todo lo que queremos, todas las actividades y servicios que queremos que vivan en nuestra aplicación, tienen que estar declaradas en nuestro archivo Android Manifest.

Si lo hacemos con nuestro wizard, automáticamente esto se da de alta en mi archivo Android Manifest. Aquí está, aquí lo puedes ver. Está activity android:name y viene .ListadoMascotas. Esta anotación punto, se da porque se simplifica en dónde exactamente está ubicada esta clase.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.anncode.mimascota">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MiMascota"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".ListadoMascotas"></activity>
    </application>

</manifest>
```

Es decir, en dónde, en qué paquete está ubicada esta clase, en qué package, tú bien podrías colocar com.anncode.mimascota.ListadoMascotas. Así lo podrías ubicar.

d:name="com.anncode.mimascota.ListadoMascotas"></activity>

En el caso de que utilizáramos paquetes diferentes con distintas actividades, que nuestro proyecto esté organizado con múltiples paquetes.

Ahora, si quiero declarar mi siguiente activity, que es lo que finalmente quiero.

Puedo colocar también com.anncode y me falta DetalleMascota, y no olvides cerrar nuestra etiqueta. Una vez cerrado, pues esto es todo lo que tenemos que hacer para crear una actividad.

Recuerda, debemos crear nuestra clase java, debemos crear nuestra layout, setear la vista y darlo de alta en nuestro archivo Android Manifest. Que con nuestro asistente Wizard de Android Studio nos hace todo el trabajo.

Declarando Views (Elementos de interfaz de usuario) por XML

Uno de los componentes principales de nuestros activities son las interfaces de usuario.

Así que te voy a enseñar a crear esas interfaces de usuario que siempre has deseado tener en la aplicación.

Primeramente te voy a mostrar como manejar los views. No solamente desde el entorno gráfico, sino también a un nivel de código xml.

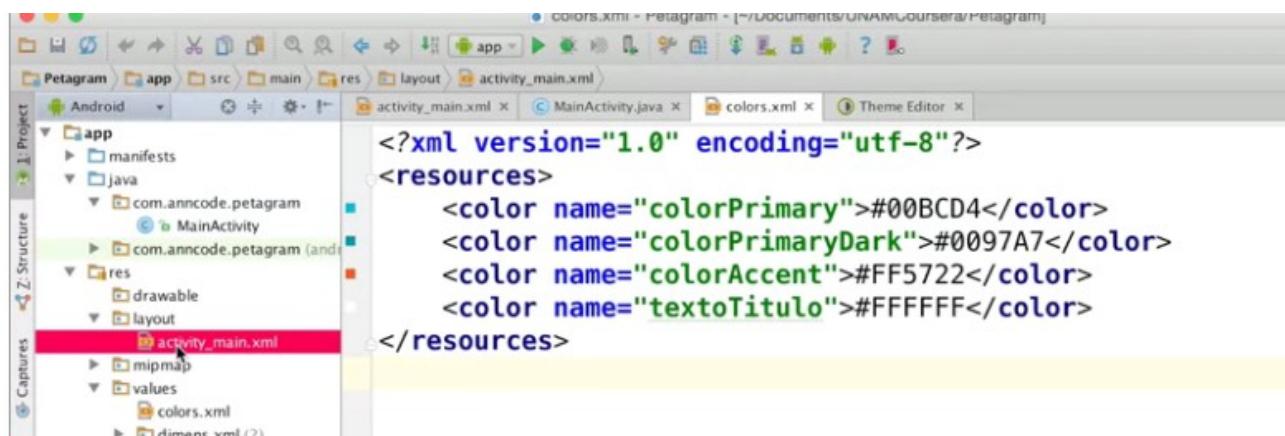
Para que evites cualquier código que de pronto se vuelva complicado para tí y no te deje moverte a tus anchas dentro del entorno.

Entonces, vamos a ver como lo hacemos desde código xml y después lo veremos también como lo vamos a hacer desde código java.

Layouts y Raised Button

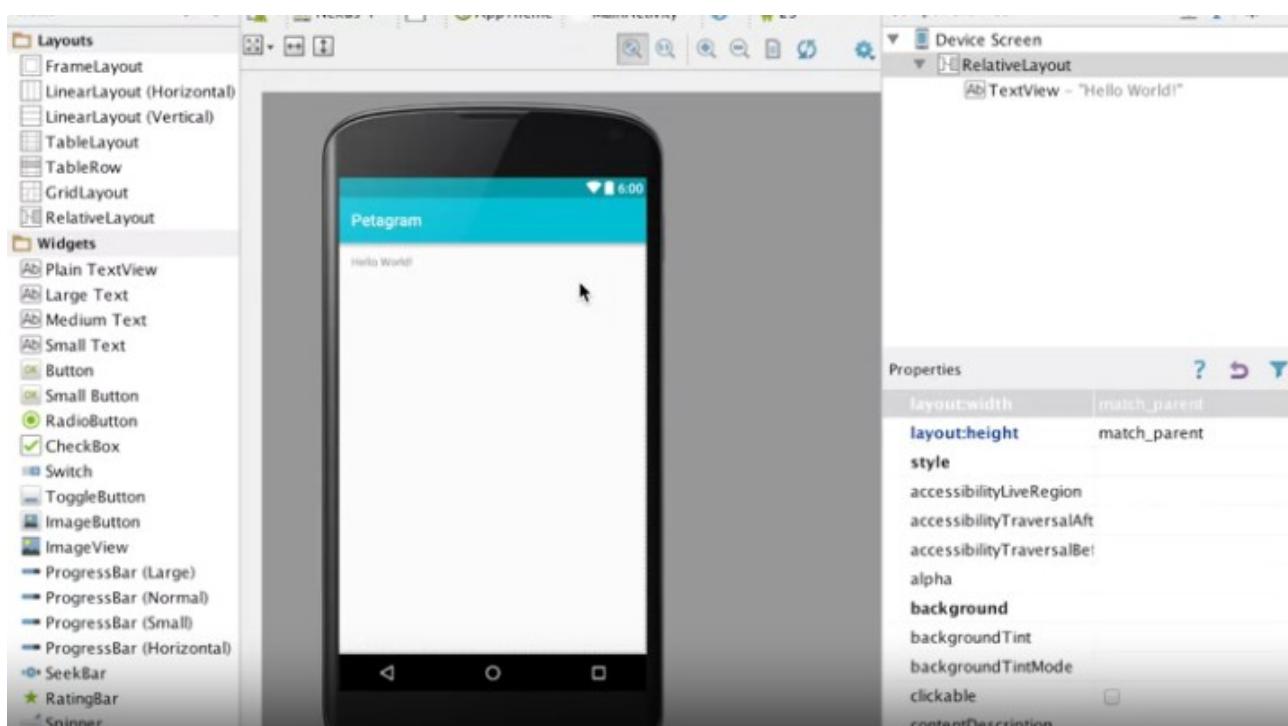
Entonces hemos creado ya algunas actividades.

Ahora vamos a ir a nuestro Layout, nuestro ActivityMain.xml. Eso lo vas a localizar en la carpeta Res, Layout y después ActivityMain.xml.



Bien. Tengo una aplicación como esta que se llama Pentagram y esta es la vista gráfica.

Este es nuestro entorno gráfico.

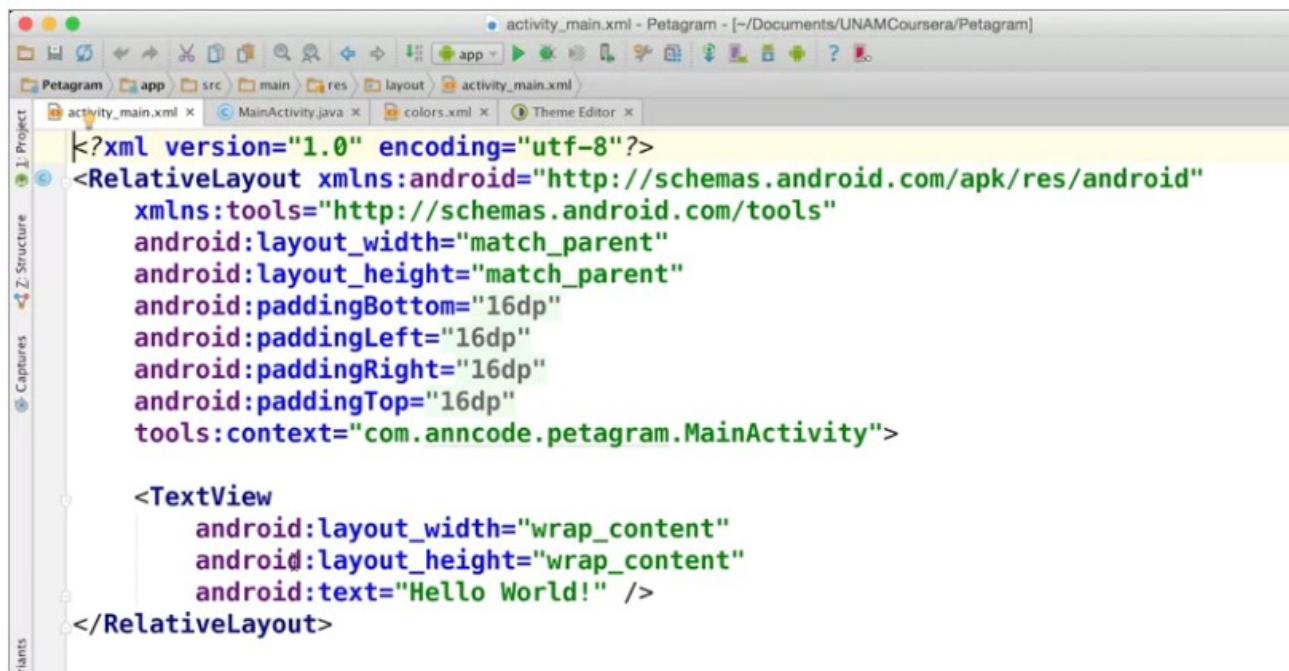


En esta parte que está aquí de este lado vamos a estar viendo todos los widgets o todos los elementos que podemos estar aquí colocando.

Y bueno, hemos visto que simplemente tocas un elemento y lo arrastras. Pero ahora vamos a verlo a nivel de código.

En la parte superior tengo dos pestañas. La primera es la vista de diseño que es en donde estoy ubicada y la segunda es la vista de texto.

Vamos a dar clic ahí en la vista de texto. Lo que vemos ahora es la vista XML.



The screenshot shows the Android Studio interface with the XML code for `activity_main.xml`. The code defines a `RelativeLayout` with various attributes and a nested `TextView` element containing the text "Hello World!".

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.anncode.petagram.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

Que cada una de estas etiquetas es lo que define lo que estamos viendo finalmente en nuestra pantalla.

Aquí de este lado yo tengo una pestaña de preview, y todo lo que tengo aquí en XML, todo esto está definido, es todo lo que se está definiendo por acá.

Entonces XML, al igual que por ejemplo HTML es un lenguaje de etiquetas. Un lenguaje de etiquetas que está estructurado a nivel de etiquetas padre y etiquetas hijo.

Y al igual que en HTML, si pudiéramos hacer una analogía, en HTML tú puedes alterar una etiqueta y puedes asignarle más estilo, puedes cambiarle el color a un texto o puedes definir un color para una etiqueta. Puedes hacer varias cosas a partir de CSS.

En Android también podemos hacer algo similar. También podemos darle estilo a estas etiquetas a través de propiedades.

Aquí como observamos tengo una etiqueta padre. Mi etiqueta padre que se llama `RelativeLayout`. Esta es la etiqueta padre que está definiendo todo este Layout.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com
    xmlns:tools="http://schemas.android.com/tools"
        ... >
```

Mas adelante vamos a ver que es un Relative Layout. Y tengo una etiqueta hijo que se llama TextView.



```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" /> I
```

Esta etiqueta hijo yo sé que es su etiqueta hijo porque está contenida dentro de la otra etiqueta, de relative layout.

De esta forma es como yo me doy cuenta cuando tengo en la jerarquía de las etiquetas.

También otra forma en que puedo darme cuenta y más que nada cómo definir una etiqueta padre, es la forma en que vamos a estar cerrando nuestras etiquetas.

Por aquí yo tengo relative layout. Aquí en realidad abre, aquí está la apertura de la etiqueta, pero al cerrarse está de este lado. Se cierra con diagonal y se está repitiendo el nombre de la etiqueta.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com
    ...>
</RelativeLayout>
```

Esa es una forma de cerrar una de nuestras etiquetas en XML.

La otra forma que tenemos es simplemente como se ve aquí, en esta etiqueta de text view. Se cierra con diagonal y pico paréntesis.



```
<TextView
    android:layout_width="wrap_content"
    ...>
/>
```

Como observas aquí yo no estoy repitiendo nuevamente el nombre del TextView, sino simplemente estoy cerrando de esta forma.

Lo hago así porque esta primera forma me esta diciendo que esta etiqueta está disponible para alojar a más views o más elementos, más etiquetas.

Entonces yo dentro de ese elemento puedo definir cuantos más elementos yo quiera. Entonces si yo uso esta notación entonces eso quiere decir que solamente este elemento o esta etiqueta nunca va a

contener otra. Nunca va a ser una etiqueta padre de otra. Sino simplemente va a ser una etiqueta hijo o una etiqueta aislada por ahí.

Entonces como veníamos hablando sobre los estilos, aquí también yo puedo asignarle una forma o puedo asignarle propiedades a mis elementos.

Lo haré a través de las propiedades que van a estar en este color como moradito que dice android: y enseguida viene la propiedad que está afectando a esa etiqueta en cuestión.

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.anncode.petagram.MainActivity">
```

Por ejemplo aquí yo estoy definiendo un layout width, es decir, este elemento de qué tamaño será su ancho. O yo puedo definir un layout height, de qué tamaño será su altura.

Además puedo decir que esa etiqueta tenga un padding en la parte de abajo, un padding en la izquierda, un padding en la derecha y además también un padding al top.

Entonces todos estos elementos están afectando aquí. ¿Y cómo lo sé? Porque bueno, aquí está cerrando mi pico paréntesis.

Esto va a ser lo que siempre vamos a estar trabajando, siempre vamos a estar trabajando en esta zona de mis XML.

Entonces bueno, aquí ya tengo un view, tengo mi primer view. Todos estos elementos, todas estas etiquetas, provienen de la clase padre view.

Y por eso siempre se van a conocer como views.

Algunos incluso ya en su nombre traen la palabra view, como este que se llama text view.

Y entonces todos los elementos, todo lo que yo vea dentro de mi interfaz, todo va a ser un view, todos esos elementos son views. O también a veces se les conoce como widgets.

Bien, entonces este text view lo que yo estoy definiendo en esta propiedad es un width.

Dentro del width y también para el height yo tengo dos parámetros que puedo decidir que ese texto, esa etiqueta del texto, que tanto sea su ancho y que tanto sea su altura.

Aquí este text view inmediatamente cuando lo selecciono aquí me lo muestra, de este lado me muestra el elemento que estoy editando.

The screenshot shows the Android Studio interface. On the left, the Project tool window is open, showing files like activity_main.xml, MainActivity.java, colors.xml, and Theme Editor. The main area displays the XML code for activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.anncode.petagram.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

</RelativeLayout>
```

The code editor highlights the `wrap_content` attribute for both `layout_width` and `layout_height`. To the right, the Preview tool shows a smartphone screen with the title "Petagram" and the text "Hello World!" displayed.

Y entonces yo tengo el primero, dice “wrap_content”.

Wrap content significa que este elemento, este view, va a ser tan ancho como el contenido de ese elemento. ¿Cuál sería su contenido? En este caso tengo aquí la propiedad texto, la propiedad text, que almacena el texto Hello World!

Entonces este Text View va a ser tan ancho como la cantidad de texto que tenga.

Como observas aquí, pues aquí se ve envuelto el texto en color azul. Se ve envuelto el texto en un color azul, esto significa que esa etiqueta es tan ancha como el texto crezca.

Si yo le agrego, por ejemplo, otro signo de admiración hacia abajo automáticamente cambia.

Si le pongo a lo mejor una carita feliz automáticamente va creciendo el elemento conforme a la cantidad de texto que tenga.

Ahora, has podido ver que tengo otro parámetro disponible, este que está aquí en mi etiqueta de relative layout.

Dice, Match Parent, va a decir. Si yo pongo aquí Match Parent, por ejemplo en el Text View.

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Hello World!" />
```

¿Qué ha pasado? Bueno, pues aquí no importa que tanto texto tenga, significa que este View va a ser tan grande como su etiqueta padre. Es decir, ajústate a la etiqueta padre.

Entonces aquí estoy diciendo que el Width va a ser tan grande como la etiqueta padre.

¿Quién es etiqueta padre? Bueno, Relative Layout. Relative Layout en este caso es todo esto de aquí, todo lo blanco que podemos ver. Todo eso es Relative Layout. Entonces la etiqueta va a ser de ese tamaño.

Por eso también el Relative Layout dice Match Parent. Aquí también dice ajústate a tu padre.

Bueno en este caso tu padre, el parent del Relative Layout es la ventana donde estamos viendo la aplicación móvil, la interfaz de usuario.

Entonces esas son las dos opciones que tengo para definir el ancho y también el alto de un elemento.

Match Parent que va a ser tan ancho como la etiqueta parent lo sea. Y Wrap Content va a ser tan ancho como el contenido de ese elemento.

Voy a dejar mi Text View como Wrap Content.

Bien, es muy importante que sepas que debemos tener siempre como propiedades obligatorias el width de un elemento y también el height de un elemento, de cualquier view.

Siempre Andorid Studio nos lo va a solicitar y siempre los debemos tener presentes, e width y el height, layout width y layout height, son distintos.

Algo que a mí también es bueno colocar siempre es una propiedad android:id.

```
<TextView  
    android:id="@+id/"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello World!" />
```

La propiedad id. Me gusta mucho colocarla porque a partir de este id es como siempre vamos a poder estar referenciando nuestros elementos ya sea desde otro recurso, desde el mismo recurso o incluso desde un archivo de Java.

Entonces yo voy a colocarle por ejemplo a este text view un id. Le puedo poner la siguiente sintaxis que yo utilizo, es muy personal.

Primeramente coloco un prefijo del elemento, le coloqué TV, que significa Text View.

Y posteriormente con CammelCase coloco el nombre del elemento que voy a identificar.

Por ejemplo, esto podría ser un título Entonces le he colocado tvTitulo.

```
    android:id="@+id/tvTitulo"
```

Si observas, a diferencia de los otros recursos que normalmente accesamos a un recurso con arroba, nombre del recurso, como por ejemplo en el padding left, aquí está @dimen/activity_horizontal_margin. Eso significa que hay un recurso en @dimen.

```
    android:paddingLeft="@dimen/activity_horizontal_margin"
```

A diferencia de este, yo estoy utilizando este símbolo, el símbolo de más. Arroba, más, id. Es muy común que a veces tengas este tipo de dudas porque también puedes colocar @id. Con @+id significa que yo en ese momento estoy creando un nuevo recurso o un nuevo identificador.

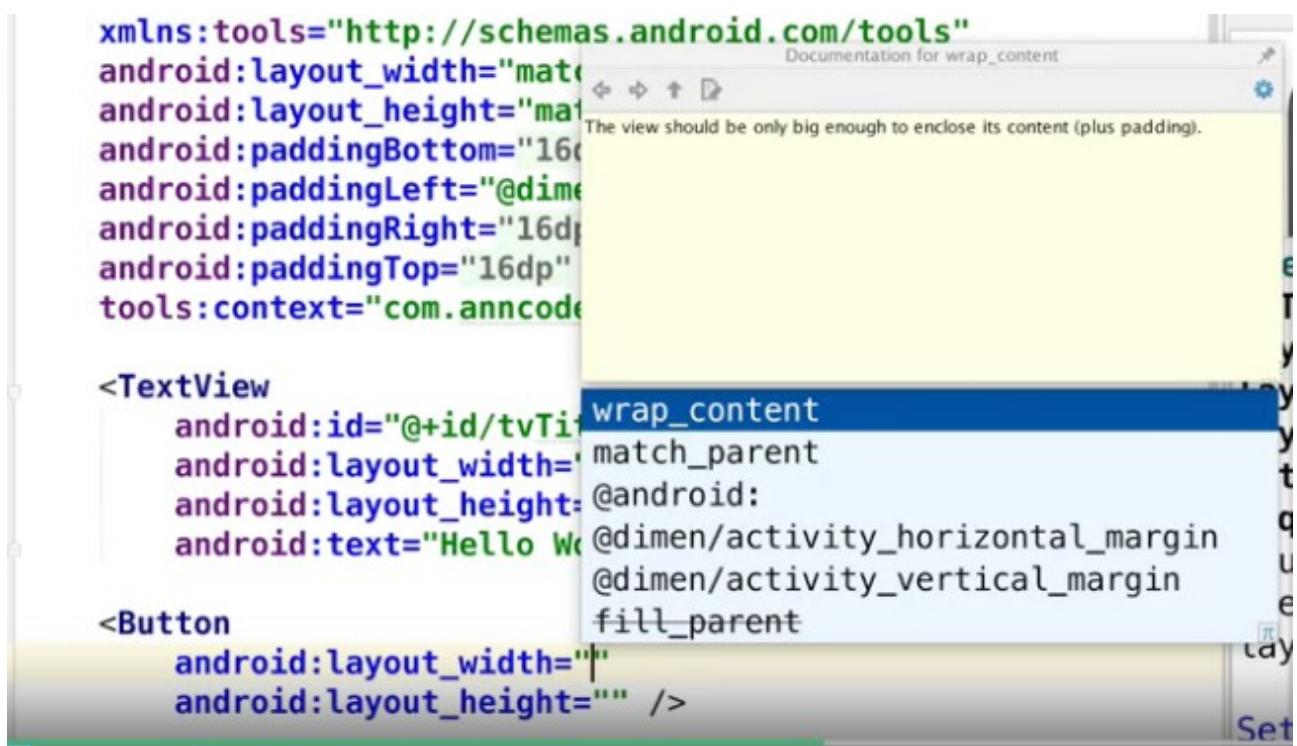
Con @+id significa que estoy añadiendo este text view, un identificador para este text view.

En este caso el identificador que se está mapeando en nuestro archivo R, será tvTitulo.

Y de esa forma, con esa referencia tvTitulo, voy a tener acceso a todos los parámetros a todos los métodos de este text view, desde código Java por ejemplo.

Entonces este es un elemento que podemos tener.

También podemos tener botones. Y simplemente para comenzar a escribir puedo colocar button. Si observas me empieza a sugerir el IDE todo lo que puedo poner.



The screenshot shows an Android XML layout file in an IDE. The code is as follows:

```
xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.anncohen.helloandroid.MainActivity"

<TextView
    android:id="@+id/tvTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

A code completion dropdown is open over the word "wrap_content" in the first TextView's height attribute. The dropdown contains several options: "wrap_content" (highlighted in blue), "match_parent", "@android:parentSelectorResId", "@dimen/activity_horizontal_margin", "@dimen/activity_vertical_margin", and "fill_parent".

Me dice aquí que si quiero que tenga un Wrap Content.

Quiero que el botón sea tan ancho como su padre, es decir, que sea un botón largo, para eso pongo el width como "match_parent". Puedo colocar Height Si le pongo al botón que sea tan largo como su padre, va a quedar un botón muy muy largo.

Entonces voy a definir wrap content para el height. Voy a definir aquí también un id para este botón.

Le voy a poner btn y le voy a poner "MiBoton".

```
<Button  
    android:id="@+id/btnMiBoton"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```

Bien, ya está ahí el botón.

Si observas el botón se está encimando en el Text View.



Más adelante vamos a ver cómo reacomodar o cómo vamos a estar acomodando nuestros elementos dependiendo de un layout específico.

Pero para que ahorita no me cause ruido voy a decirle que este botón esté debajo de layout below.

Utilizando esta propiedad le estoy diciendo que este elemento esté debajo de otro elemento.

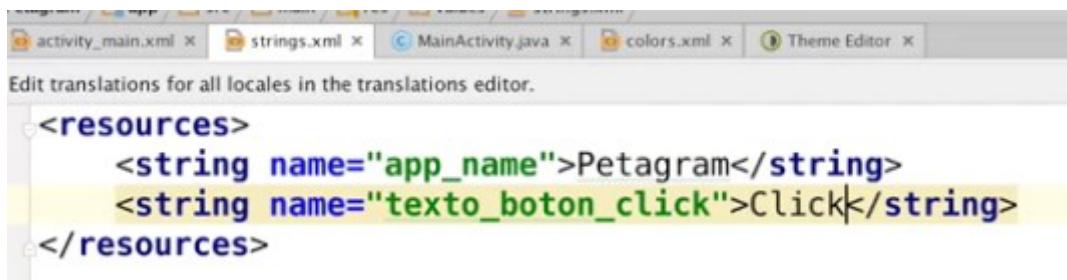
```
<Button  
    android:id="@+id/btnMiBoton"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/tvTitulo"  
    />
```

Que aquí lo que está recibiendo es un id. Entonces le pongo @+id y puedo identificar mi elemento anterior. Entonces a través del id yo puedo estar haciendo referencia a mis elementos incluso dentro de este mismo recurso que es el layout.

Bueno, ahora yo puedo ponerle también un texto al botón. Puedo colocar la propiedad Text, y puedo colocar, por ejemplo, utilizando el archivo de recurso String, le puedo colocar el App Name.

```
<Button  
    android:id="@+id/btnMiBoton"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/tvTitulo"  
    android:text="@string/app_name"  
/>
```

Pero vamos a nuestro archivo strings para definir un nuevo recurso que diga a lo mejor, vamos a ponerle aquí texto_boton que diga siguiente, o que diga click nada más.



Texto_boton_click. Click y yo le puedo poner click.

Entonces no olvides guardar tu archivo strings para que se mapee esto mejor. Y entonces aquí yo puedo poner el recurso que acabo de dar de alta, texto_boton_click.

Ahí está. Y automáticamente se coloca el texto click en mi botón.

```
<Button  
    android:id="@+id/btnMiBoton"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/tvTitulo"  
    android:text="@string/texto_boton_click"  
/>
```

Igualmente para este hello world vamos a quitarlo de aquí y vamos a hacerlo de la forma correcta.

```
<resources>  
    <string name="app_name">Petagram</string>  
    <string name="texto_boton_click">Click</string>  
    <string name="texto_titulo">Hello World!</string>  
</resources>
```

```
<TextView  
    android:id="@+id/tvTitulo"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/texto_titulo" />
```

Estamos viendo cómo manejar todas nuestras vistas.

Texto título, y siempre debemos tener los strings dentro de este archivo, "Strings.xml", para que esté mejor organizado nuestro proyecto y además sea muy sencillo también poder lograr una traducción en nuestra aplicación.

Entonces ya tengo mi segundo view, un botón, ahí está.

Pero dentro de Material Design tenemos nuevos estilos y nuevos tipos de botones que hay disponibles. Recuerdas nuestro botón Raise Button. Nuestro botón Raise Button no es un botón como este, sino es un botón un poco más lindo, que tiene más profundidad, que cuando lo clickeas el botón se levanta.

Que puede tener un color definido y cuando haces touch, cuando haces tap en el botón.

Puede cambiar de color o puede verse el efecto de agua, de gota de agua dentro del botón.

Entonces, para esto, para poder darle este estilo a mi botón, como un botón raise button de Material Design. iremos a nuestro archivo styles, vamos a nuestro archivo styles. Y lo que voy a hacer dentro de styles es definir un estilo para mi botón.

Un estilo que va a llevar el estilo precisamente de Material Design.

Entonces voy a comenzar declarando un Style, le puedo, aquí en la etiqueta Name, en el atributo Name, es la forma en como voy a identificar ese estilo.

Entonces le puedo colocar MiRaisedButton. Mi botón. Vamos a poner MiBotonRaised.

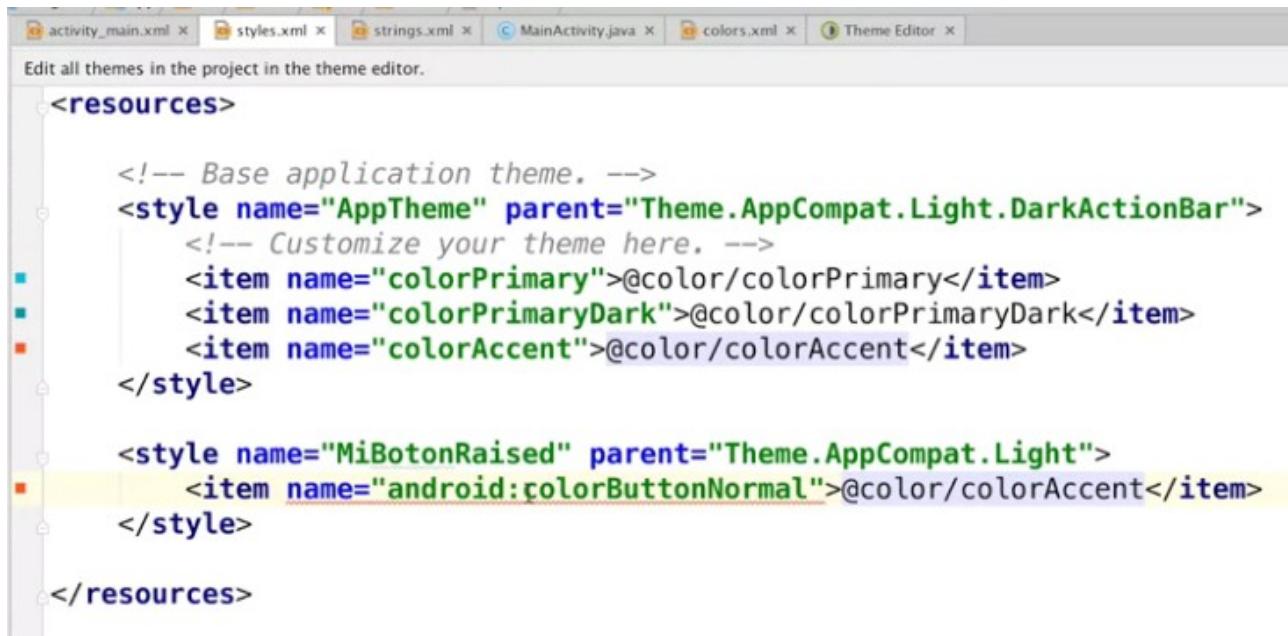
Ahí está. Este va a ser nada más el identificador para estar llamando ese estilo.

Luego viene Parent. Aquí vamos a definir el estilo "Theme.AppCompat.Light".

Con esto cerramos nuestra etiqueta y estamos definiendo un estilo para un View en específico. Esto lo podemos hacer en la medida de lo posible siempre que queramos.

Entonces voy a empezar ahora a alterar, o voy a configurar algunas de las propiedades de este tema que afecten a mi botón. Por ejemplo, puedo colocar el color, Color Button Normal.

Puedo decir como quiero que sea el color de mi botón, de un botón normal. Ahorita el botón es de color gris, no me gusta. Voy a decirle que tome como recurso, del archivo Colors, que tome el color Accent.



```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="MiBotonRaised" parent="Theme.AppCompat.Light">
        <item name="colorButtonNormal">@color/colorAccent</item>
    </style>

</resources>
```

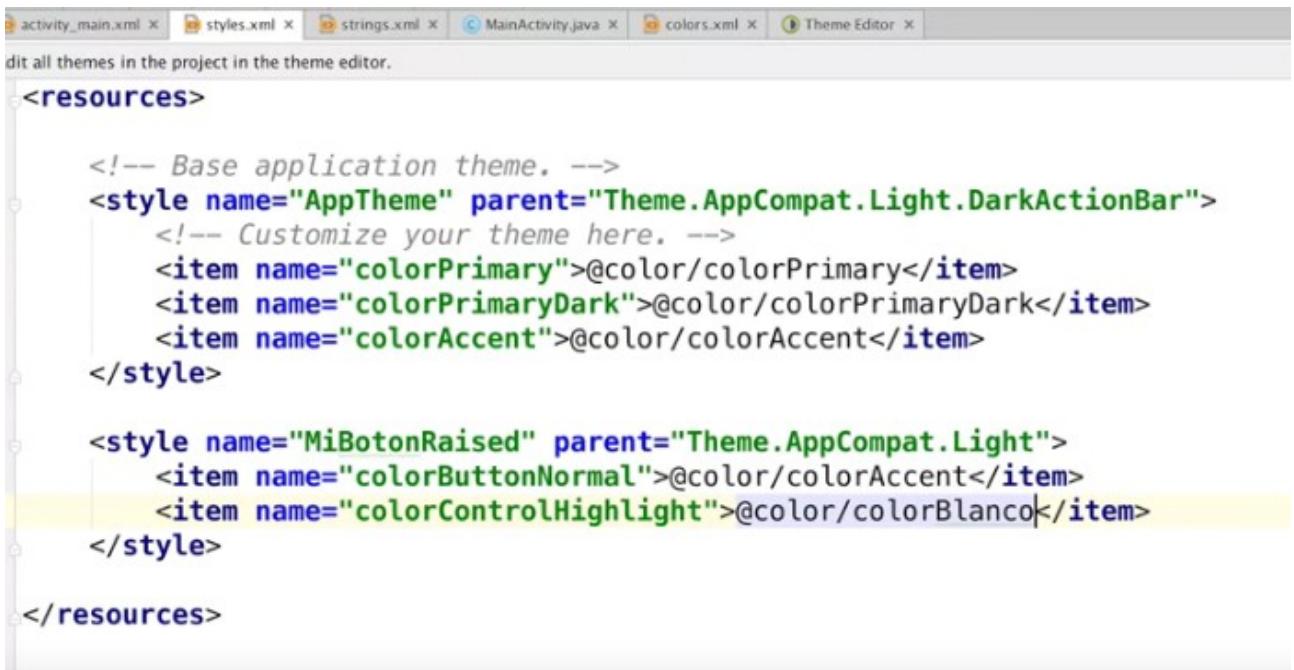
Voy a quitarle esos a "Android:"

```
<style name="MiBotonRaised" parent="Theme.AppCompat.Light">
    <item name="colorButtonNormal">@color/colorAccent</item>
</style>
```

Ahora voy a colocar otra propiedad, otra configuración cuando mi botón esté presionado.

Cuando mi botón esté presionado recuerda que se hace un efecto de agua. Entonces yo podría decir que cambie un poco de color o que se vean las ondas del agua que está atravesando en el botón con otro tono que sea distinto para que se vea más el efecto.

Para hacer esto puedo poner "ColorControlHighLight".



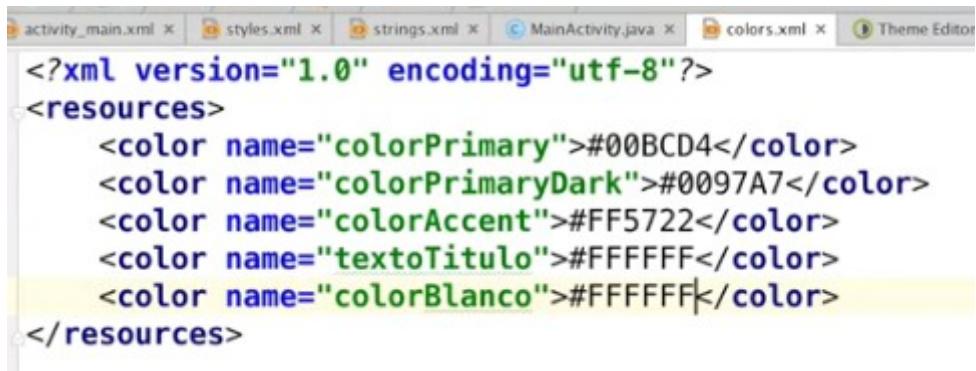
```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="MiBotonRaised" parent="Theme.AppCompat.Light">
        <item name="colorButtonNormal">@color/colorAccent</item>
        <item name="colorControlHighlight">@color/colorBlanco</item>
    </style>

</resources>
```

Entonces puedo poner Color. Le puedo decir que a lo mejor quisiéramos que fuese un color blanco. Y puedo ir a mi archivo Colors a definir un color blanco.



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#00BCD4</color>
    <color name="colorPrimaryDark">#0097A7</color>
    <color name="colorAccent">#FF5722</color>
    <color name="textoTitulo">#FFFFFF</color>
    <color name="colorBlanco">#FFFFFF</color>
</resources>
```

Ahí. Puedo colocarlo también en color blanco, en mi archivo style, simplemente indicándolo el hexadecimal.

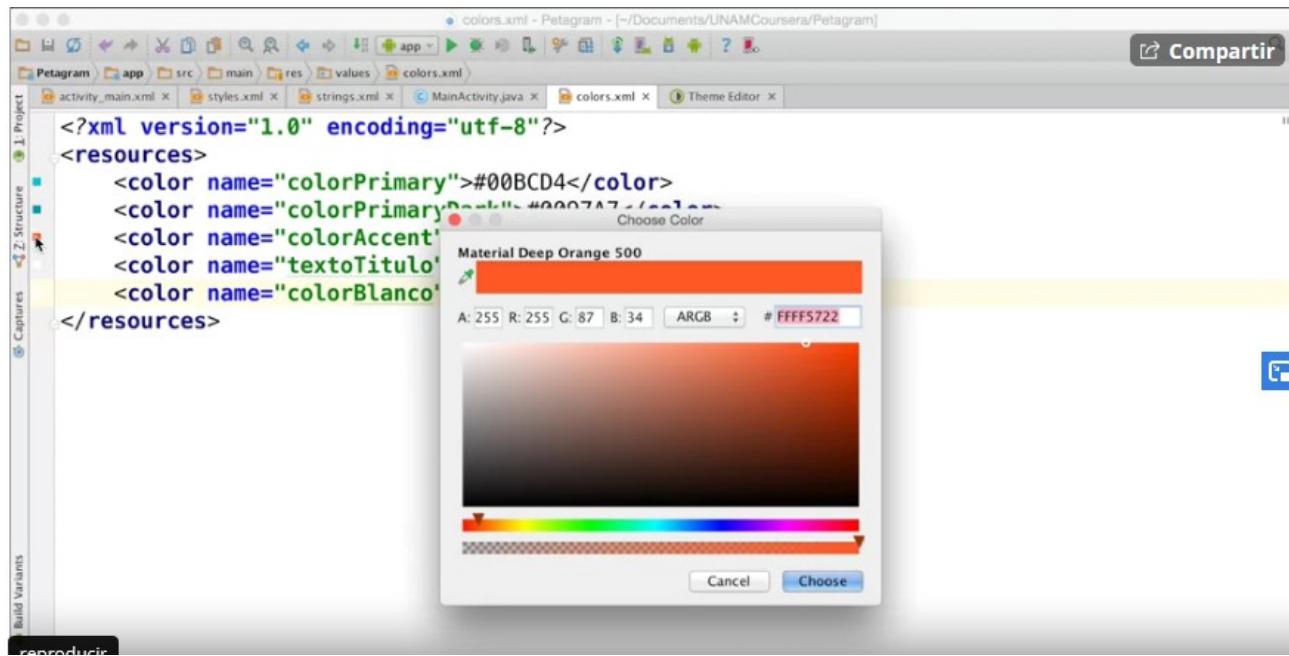
Pero la verdad es que es mucho mejor tener todo organizado tal cual como debe ser.

Le voy a poner color blanco, así para tener siempre identificado este elemento. Color blanco y regresaré a mi Styles y voy a colocar @color/colorBlanco.

Entonces en mi archivo Colors, he dado de alta un nuevo recurso que se llama ColorBlanco. Le puse el hexadecimal.

Y luego, en Styles, he hecho referencia a ese color blanco.

Cuando en nuestro archivo Colors podamos tener un Picker de los colores que tenemos disponibles.



Entonces por acá puedo también definir el color de texto del botón.

O recuerda que esto se parece un poco a CSS si han tenido la oportunidad de trabajar con esto.

Entonces puedo poner textColor. Si han tenido la oportunidad de trabajar con CSS se darán cuenta de que en CSS es muy común estar definiendo estilos y a ese estilo definirle el color o cada propiedad que deba tener y finalmente se aplica el estilo.

Entonces yo en styles.xml le voy a decir que el textColor de este botón también sea un color blanco.

```
<style name="MiBotonRaised" parent="Theme.AppCompat.Light">
    <item name="colorButtonNormal">@color/colorAccent</item>
    <item name="colorControlHighlight">@color/colorBlanco</item>
    <item name="textColor">@color/colorBlanco</item>
</style>
```

Un color blanco que voy a tomar del recurso. Y aquí vamos a poner nuevamente @color. Y voy a poner colorBlanco.

Aquí automáticamente, ésto es lo que he configurado para ese botón, ¿sí?, y como estoy definiendo para este estilo que tenga el estilo de Material Design.

Pues los botones, el Material Design, este tipo de botones, son botones Raised, botones elevados.

Entonces ahora vamos a aplicar este estilo a nuestro botón. Voy a poner Android Theme. Con esa propiedad, Android Theme. Y voy a poner @style/ y colocamos nuestro estilo que pusimos.

```
<Button  
    android:id="@+id/btnMiBoton"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/tvTitulo"  
    android:text="@string/texto_boton_click"  
    android:theme="@style/MiBotonRaised"  
/>
```

Entonces puedo poner aquí nuestro estilo Botón Raised. Y vemos que el botón ya ha cambiado, ahora tiene otro estilo. Vamos a correr esto.

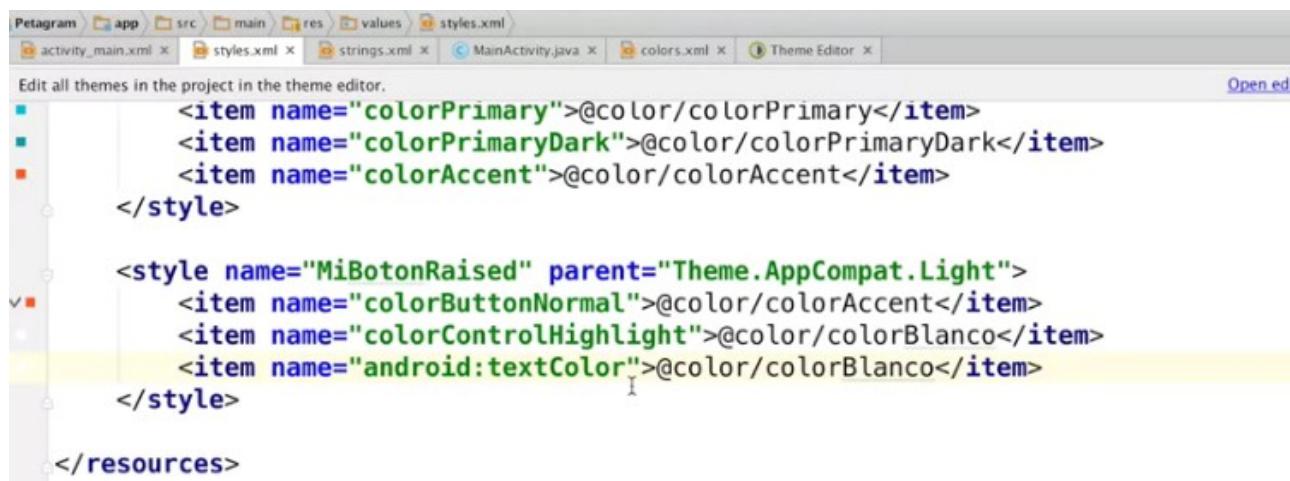
Yo ya tengo previamente conectado un dispositivo Android a mi computadora.

Previamente también ya he instalado todos los drivers necesarios que todo lo que necesite mi computadora necesite para conectarse a mi dispositivo.

En el caso de Windows, y sobre todo en dispositivos de marcas no tan conocidas. Generalmente el IDE Android Studio no reconoce nuestro dispositivo. Entonces lo que vamos a hacer es instalar todos, todos, absolutamente todos los drivers. Y cuando nosotros corremos nuestro proyecto aquí ya debe aparecer nuestro dispositivo conectado.

Por aquí me está diciendo que tengo un error.

Dice que no reconoce eso, vamos a poner la propiedad android:textColor.



```
<resources>  
    <item name="colorPrimary">@color/colorPrimary</item>  
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>  
    <item name="colorAccent">@color/colorAccent</item>  
    </style>  
  
    <style name="MiBotonRaised" parent="Theme.AppCompat.Light">  
        <item name="colorButtonNormal">@color/colorAccent</item>  
        <item name="colorControlHighlight">@color/colorBlanco</item>  
        <item name="android:textColor">@color/colorBlanco</item>  
    </style>  
</resources>
```

Y aquí, ahora sí you, de hecho ya se alcanza a ver. Se alcanza a ver ya las letras blancas.

Perfecto.

Entonces vamos a esperar a que Gradle empiece a construir nuestro proyecto. Empiece a unificar todo lo que acabamos de hacer y que nos muestre ya no tenemos ningún error.

Aquí está nuestro botón. Y como observas, pues al yo seleccionarlo, al darle tap, como que el botón se ilumina. Como que de repente el botón se ilumina y en este caso se ilumina con un color blanco.

De hecho si eres un poco observador se ve cómo el botón se eleva, se eleva de la superficie. Y entonces esto, esto es un botón Raise. Observas todos estos taps que son como si estuviera en el agua.

Floating Action Button

Bien, ya hemos creado un Raised Button, ahora vamos a crear un Floating Action Button.

Los Floating Action Button son estos botones circulares que normalmente están en la parte inferior derecha de las aplicaciones, este a veces, generalmente es su lugar, pero en realidad podrían estar en cualquier parte, en cualquier zona que a ti te agrade o en cualquier zona que realmente lo veas útil.

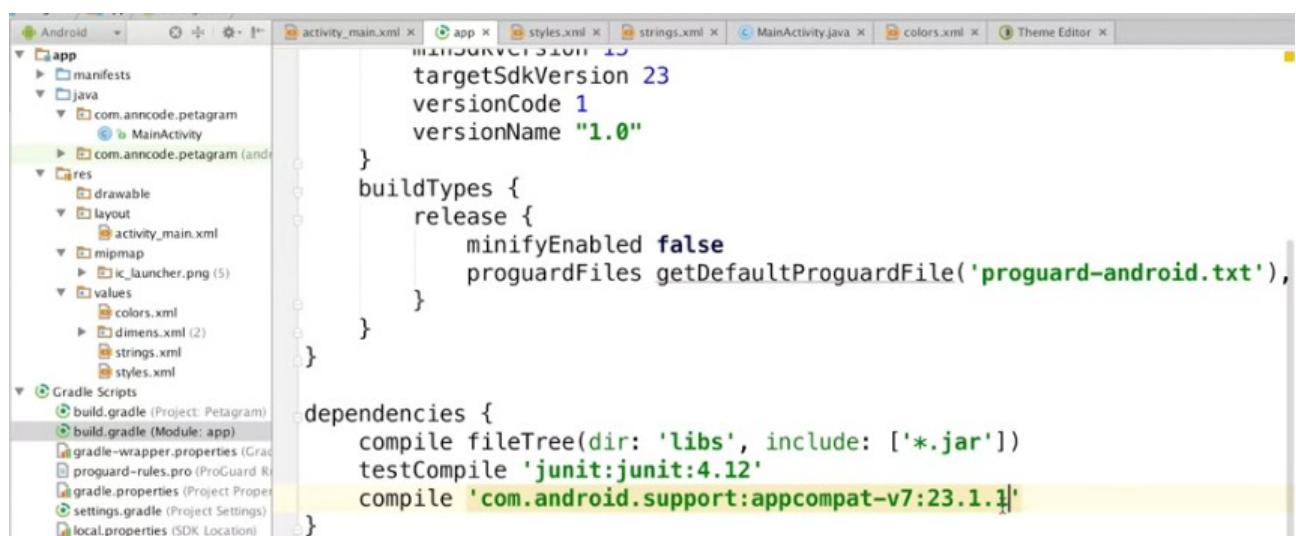
Bien, vamos a hacer uno de estos y lo primero que te debo decir es que para poder incrustar este tipo de view, estoy aquí en mi layout, activity_main, voy a pasarme a la vista de código, a la vista de texto.

Para incrustar un botón como estos, pues no tenemos una etiqueta como tal, Floating Action Button se ha integrado dentro de material design y se ha integrado como un recurso de soporte para poderlo utilizar en versiones anteriores a Lollipop.

Se ha integrado como un recurso de soporte y bueno, esto quiere decir que tendremos que descargar una librería extra, una librería extra que será nuestra librería de diseño.

Entonces, para poder ver cómo voy a, cuál es la librería que voy a descargar.

Bueno, primero vamos a nuestro archivo Gradle, el que dice Module app. Abrimos nuestro archivo Gradle y en la parte de dependencias hasta el último, ahí aparece nuestro, las dependencias que ahorita tenemos. Dice compile y esto tiene esta librería añadida.



The screenshot shows the Android Studio interface with the project structure on the left and the build.gradle file open in the main editor. The build.gradle file contains the following code:

```
targetSdkVersion 23
versionCode 1
versionName "1.0"

buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'),
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
}
```

Nosotros tenemos que añadir otra librería aquí para que entonces ya tengamos accesible como widget, como nuevo view, nuestro Floating Action Button y muchos otros elementos más de material design.

Y que además este botón se vea, no solamente en Lollipop o Marshmallow, sino en versiones anteriores a ellos.

Vamos al sitio, vamos al sitio de Google, de Android. Android es developer.android.com. Vamos para allá, y vamos a darle clic donde dice Develop.

<https://developer.android.com/topic/libraries/support-library/revisions>

<https://developer.android.com/topic/libraries/support-library/rev-archive>

En Develop, vamos a ir a Build your first app. Y bueno, tenemos esto, ¿no? Build your first app.

Entonces voy a darle clic aquí donde dice Herramientas, y del lado izquierdo, en esta barra que está aquí, vamos a ir al menú que dice Support Library.

En ese menú, en ese link, vamos a encontrar todas las librerías de soporte que podemos incluir en nuestros proyectos de material design o de Android.

Entonces, vamos a ir a la parte de Features, aquí en Support Library.

En Features, aquí están todas las actualizaciones, sobre todo los updates menores que ha tenido Android en general, algunos más grandes que otros.

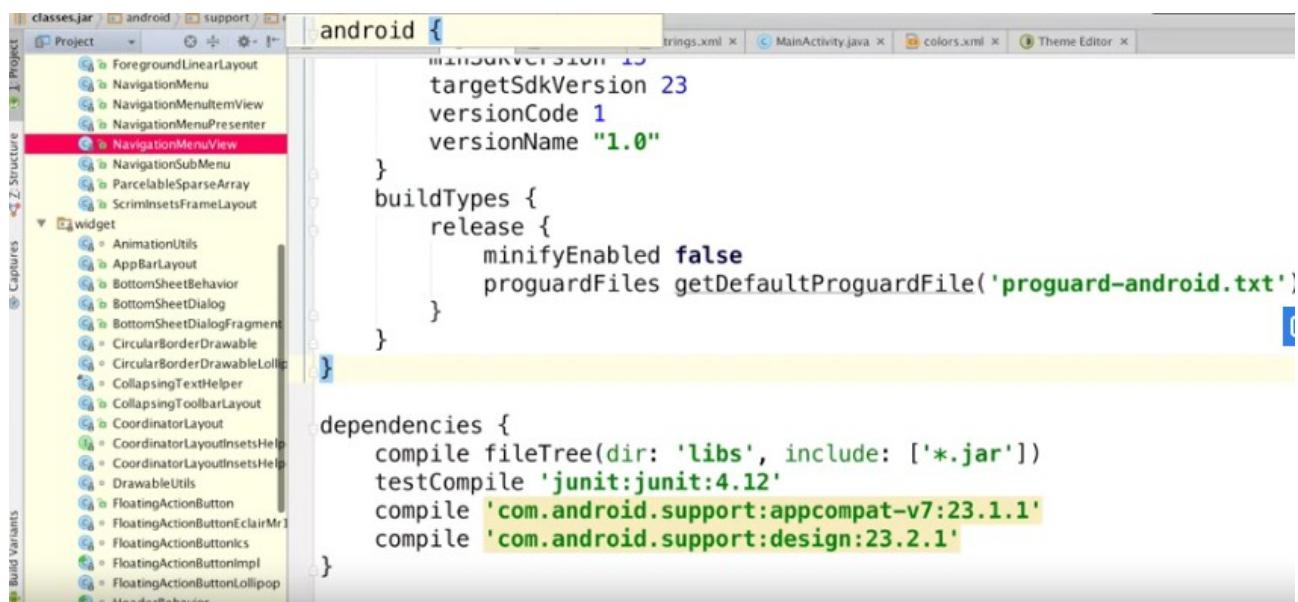
Y entonces pues por ejemplo, pues aquí está la versión 4 de Support Library y más abajo, pues está una versión 7 de Support Libraries y aquí dentro podemos ver todo lo que se ha integrado: un ActionBar, un AppCompatDialog, un cardview library, etc.

El que nosotros buscamos, es el de diseño, que para mí es más fácil ubicarlos de este lado del documento, en el lado derecho, y aquí está el menú que necesitamos Design Support Library.

Te comarto este enlace porque pues bueno, cualquier otro elemento que tú necesites, pues aquí van a venir los paquetes exactos, los paquetes que tenemos, que debemos colocar.

Entonces pues, como observas, este es el que debemos copiar, com.android.support: design.

Lo voy a copiar y esto mismo es lo que voy a pegar acá en mis dependencias.



```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.3.1'
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

dependencies {
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.android.support:design:23.2.1'
}
```

Voy a escribir compile, voy a poner dos comillitas simples y voy a darle control v.

Al darle control v, y al momento, en este momento, pues en Gradle nos aparece, ha detectado que ha habido cambios en nuestro proyecto y entonces, nos dice que debemos sincronizarlo.

Vamos a sincronizarlo para que en este momento, lo que Gradle hará será ir a Internet y descargar todo este paquete de diseño y tenerlo accesible e integrado dentro de nuestra computadora, dentro de nuestro proyecto.

Ajá, ¿cómo puedo saber si Gradle ya hizo este trabajo, si ya terminó? Bueno, en la parte inferior, aquí aparece, pero también puedo ver acá en esta pestaña, puedo cambiar la vista a Proyecto.

Y en la vista de Proyecto, aquí aparece, este es el sistema de archivos que realmente ha creado en nuestra computadora y acá está External Libraries, si abrimos eso, aquí vamos a ver todas las librerías externas que nuestro proyecto está utilizando.

En este momento ya vemos integrada la librería de diseño.

Y entonces, si eres un poco curioso y te metes un poco más a todo lo que está aquí adentro, pues puedes ver que aquí están todas las clases.

Por ejemplo, en widget, aquí están todos los elementos que ahora tengo disponibles. Y aquí está, ahora ya mi Floating Action Button, ya lo voy a poder integrar.

Perfecto, voy a colocar, voy a regresar a la vista de Android para que todo esté más presentable.

Entonces, una vez ya hecho esto en Gradle, una vez que ya se sincronizó, ya descargó todo y verificaste que la librería de diseño ya está integrada en el proyecto, ahora sí podemos empezar a colocar nuestra etiqueta de Floating Action Button.

Voy a colocar la etiqueta y aquí tengo que colocar `Android.support.design`.

`Widget.FloatingActionButton` y aquí ya aparece sugerido.

The screenshot shows the Android Studio XML code editor with the file `activity_main.xml` open. The code defines a layout with three main components:

```
    android:id="@+id/tvTitulo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/texto_titulo" />

<Button
    android:id="@+id(btnMiBoton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/tvTitulo"
    android:text="@string/texto_boton_click"
    android:theme="@style/MiBotonRaised"
    />

<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Recuerda que nos pide los atributos obligatorios.

Vamos a colocarle `wrap_content` y `wrap_content` en `width` y en `height`.

¿Qué tenemos en la vista de preview? Todavía nada, no tenemos todavía nada en la lista de preview. De hecho, dice que hay problemas de rendering.

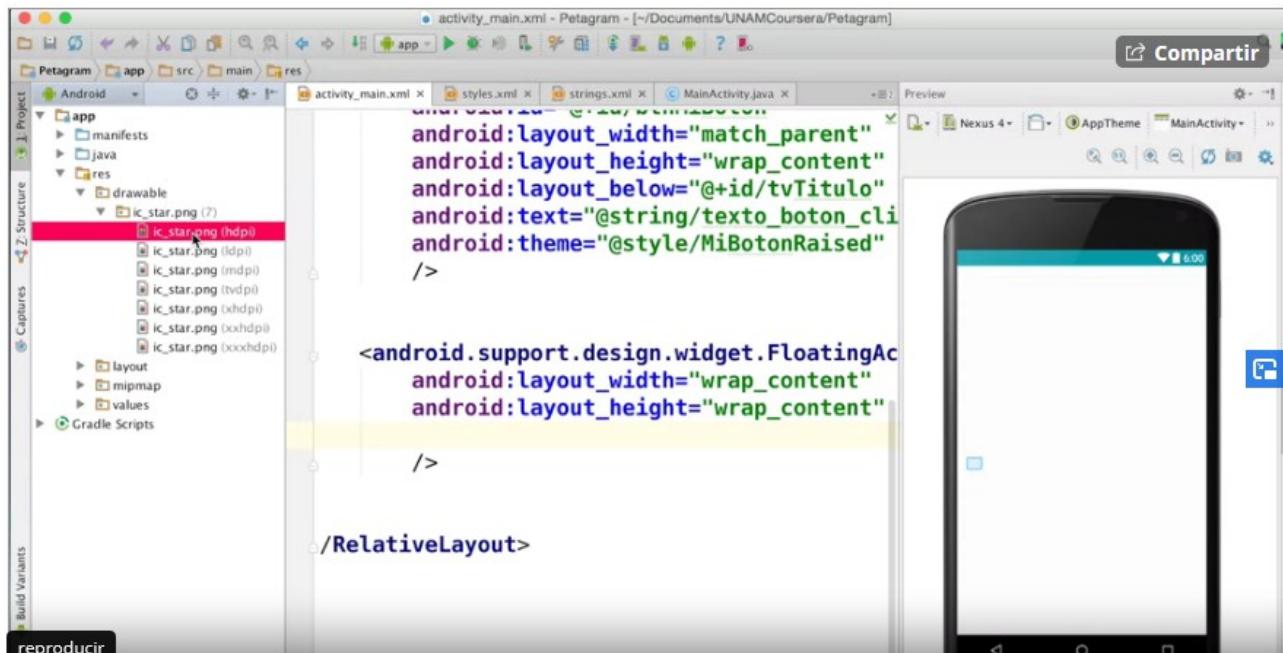
Bueno, no hay problema. Vamos a continuar. Podemos colocarle a este elemento, un ícono. Podemos colocarle un ícono de una estrella.

Entonces, el ícono de la estrella yo ya lo tengo por acá listo.

Tengo esta `ic_star` que bueno, yo ya tengo listo ese ícono, así que lo voy a traer a mi proyecto en todas las dimensiones que necesite.

Voy a darle clic derecho y recuerda nuestro plugin que insertamos y entonces, voy a hacer esta estrella en todas las dimensiones necesarias.

Aquí está, `ic_star`. Y esto como es 48, es el `medium dp`, y ya nos ha generado la estrella, en este caso en la carpeta `drawable`.



Recuerda que todos los íconos, todos los íconos de la aplicación, puedes deben estar en la carpeta Mipmap.

Pero esta ocasión la vamos a dejar ahí en drawable porque es una imagen que vamos a estar ocupando por ahí.

Entonces, voy a colocar aquí android:src y vamos a @drawable y vamos a colocar nuestra estrella.

```
<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_star" />
```

Perfecto, ahí está nuestra estrella y vamos a darle una posición donde debe estar ubicada.

Vamos a decirle que esté below de nuestro botón de arriba, de MiBoton.

```
<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_star"
    android:layout_below="@+id/btnMiBoton" />
```

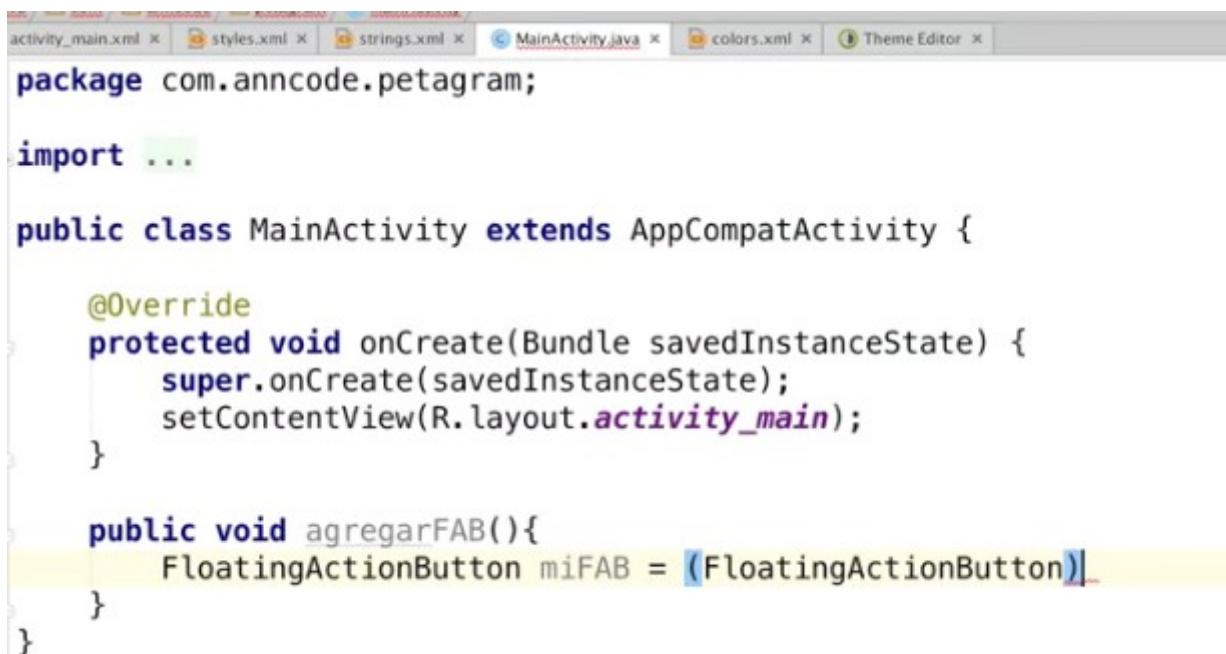
Vamos a definirle también un id, android:id, @+id y le voy a poner, eso se va a llamar fabMiFAB.

```
<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_star"
    android:layout_below="@+id/btnMiBoton"
    android:id="@+id/fabMiFAB"
/>
```

Vamos a la lista de diseño, nos dice que hay problemas de rendering. Vamos a darle refresh, vamos a esperar a que refresque todo y vamos a cambiar a lo mejor de API.

Bien, entonces ya hemos añadido nuestro botón, tenemos ahí añadido nuestro botón. Ahora, tenemos que hacer un ajuste adicional.

Vamos a nuestro archivo de Java, a nuestro main activity, el archivo quien está controlando este layout y tenemos que también agregar este elemento con código.



```
activity_main.xml x styles.xml x strings.xml x MainActivity.java x colors.xml x Theme Editor x
package com.anncode.petagram;

import ...

public class MainActivity extends AppCompatActivity {

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

    public void agregarFAB(){
        FloatingActionButton miFAB = (FloatingActionButton)
    }
}
```



```
miFAB = (FloatingActionButton) findViewById(R.id.fabMiFAB);
```

Entonces, voy a hacer por acá un método, public void voy a poner agregarFAB, ¿sí?

Y como ya tengo la librería de diseño importada, pues ya puedo tener acceso a la clase.

Y entonces vamos a poner por aquí un clásico FindViewById, donde vamos a estar llamando a nuestro botón, que tenemos allá definido en nuestro layout.

Este no es, mi botón no es. Se llama fab, este sí es.

Perfecto, ahora sí lo reconozco.

Y entonces vamos a hacer que, pues bueno, en el momento en el que alguien haga clic, a este botón, vamos a ponerle un SetOnClickListener, que esté preparado para ejecutar algo. Vamos a poner View.OnClickListener y en ese momento empieza a importar todo.

```
public void agregarFAB(){
    FloatingActionButton miFAB = (FloatingActionButton) findViewById(R.id.fabMiFAB);
    miFAB.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            ...
        }
    });
}
```

Entonces, ¿qué fue lo que hice? Tal vez esto sucedió muy rápido, simplemente le puse .setOnClickListener, buscamos este método. Este método siempre va a estar escuchando cuando nosotros le demos clic, o siempre va a estar alerta para que cuando nosotros le demos clic ejecute lo que esté a continuación.

Este método recibe un view como, como parámetro, entonces vamos a aquí, ya nos está sugiriendo esta interfaz como ClickListener, y automáticamente pues nos sobreescribe todo lo que está aquí añadido.

Entonces cuando alguien le de clic a este botón, pues va a suceder algo.

Va suceder algo por ahí. Bien, entonces este método agregarFAB, lo voy a colocar, lo voy a estar llamando, en mi método onCreate.

```
public class MainActivity extends AppCompatActivity {

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

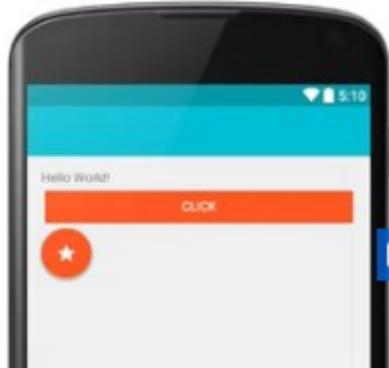
    public void agregarFAB(){
        FloatingActionButton miFAB = (FloatingActionButton) findViewById(R.id.fabMiFAB);
        miFAB.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ...
            }
        });
    }
}
```

Entonces vamos a dejarlo ahora hasta ahí, vamos a correr esto a ver cómo se ve.

Ya está casi terminando Gradle y como observas, pues bueno ya terminó, ya está compilando todo nuestro proyecto y ya aparece aquí nuestro botón.

Ya aparece, no te preocupes si no aparecía, probablemente sí aparecía para algunos, no te preocupes, aquí está, este es nuestro Floating Action Button y está aquí disponible.

Ahora vamos a correr esto mismo con nuestro emulador para tener la vista previa.



Como observas es un botón circular, los botones Floating Action Button están diseñados para que tu dedo quepa perfectamente en él y puedas darle clic de una forma super sencilla.

Y bueno pues son botones de acciones, son botones de acciones que por default puedes observar que se coloreó con el color de acentuación, se ha colocado con el color de acentuación y esto significa, porque pues es un botón de acción, es un botón que va a ejecutar acciones.

Entonces pues como ves, también acá en nuestro emulador, ahí está disponible y si tú corres estos elementos, si tú corres este proyecto en una versión anterior a Lollipop, esto debe correr muy bien, esto se debe ver perfectamente, y nuestro botón ahí está.

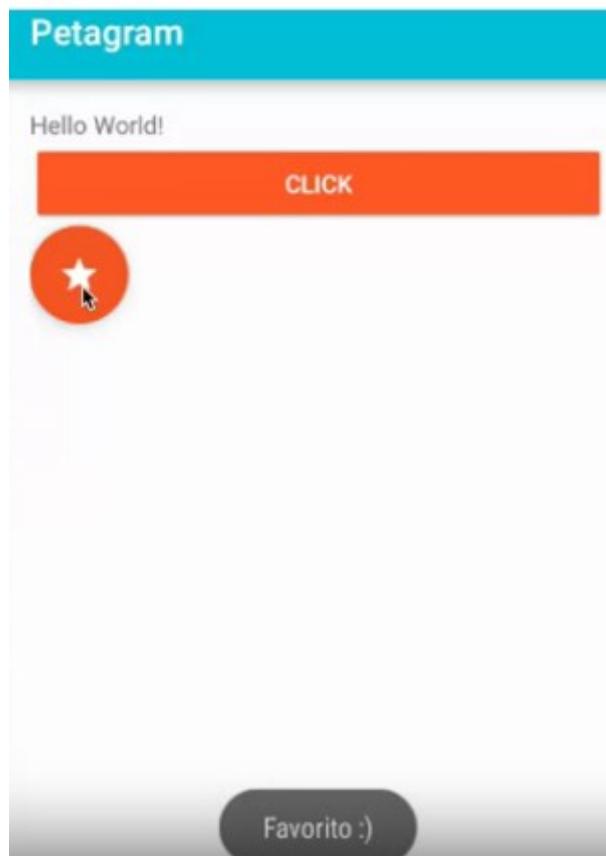
Incluso si observas también el botón Floating Action Button se está elevando, y este sería nuestro botón Floating Action Button.

Snackbar

A nuestro botón FloatingActionButton le he colocado un mensaje que dice favorito, un mensaje que cuando lo pulsas aparece así.

Estos mensajes se llaman Toast o Toast, así también son conocidos.

Aquí también lo podemos ver en nuestro emulador aquí dice favorito, favorito porque es un ícono de una estrellita.



Y aquí esta el código, este es el código que he utilizado para definir un mensaje de tipo Toast, básicamente seleccionamos la clase, definimos la clase.

```
public void agregarFAB(){
    FloatingActionButton miFAB = (FloatingActionButton) findViewById(R.id.fabMiFAB);
    miFAB.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(getApplicationContext(), getResources().getString(R.string.mensaje), Toast.LENGTH_SHORT).show();
        }
    });
}
```

Utilizamos un método `makeText`, posteriormente colocamos el contexto y el siguiente parámetro que nos recibe es el mensaje, el mensaje a mostrar o el mensaje que he definido está en mi archivo `string`, y si vamos para el `string` aquí esta una variable que se llama `mensaje` que contiene el texto, favorito, carita feliz.

```
<resources>
    <string name="app_name">Petagram</string>
    <string name="texto_boton_click">Click</string>
    <string name="texto_titulo">Hello World!</string>
    <string name="mensaje">Favorito :)</string>
</resources>
```

Lo estoy obteniendo a partir de la instrucción getResources.getString,

De esta forma es como obtenemos recursos desde código Java.

getResources.getString y gracias a nuestro archivo R.string obtenemos el mensaje.

El siguiente parámetro es nuestro, la duración del mensaje. Aquí estamos definiendo una duración del mensaje de tipo short, una duración corta que, simplemente aparece y desaparece.

```
getResources().getString(R.string.mensaje), Toast.LENGTH_SHORT)
```

Esos tipos de mensaje es que son muy útiles, son muy comunes, cuando manejamos notificaciones.

Notificaciones cuando queremos notificarle al usuario que algo sucedió exitosamente. Y entonces finalmente ya que hemos hecho todo esto, colocamos el método show, sin este método no se muestra el mensaje aunque lo hayamos configurado a la perfección. Sin show, no vamos a mostrar el mensaje.

Pero te tengo una mala noticia, estos mensajes llamados Toast ya no están disponibles o ya no entran dentro del material design sino que ahora ha llegado un nuevo elemento, un nuevo widget, que ha venido a reemplazar estos mensajes Toast.

Este widget se llama Snackbar. Entonces simplemente voy a comentar esta línea de código que tenemos en Toast.makeText y bueno cabe mencionar que me encuentro dentro del método que definimos en la sección.

Agregar FAB y estoy dentro de mi método onClick y ahí es donde cuando alguien de clic a este botón pues debe aparecer nuestro mensaje.

En este caso aparecerá un mensaje, elemento Snackbar que esos aparecen en la parte inferior de nuestro teléfono, en la parte inferior y aparecen también a modo de notificación, simplemente entran de abajo hacia arriba o simplemente aparecen y están en esta zona de nuestro teléfono.

Este Snackbar viene a reemplazar los mensajes Toast y para hacer esto, vamos a hacer uno sencillo.

Previamente yo debo tener ya integrada en Gradle, mi librería de soporte Android.support.design. para que puedas utilizar la clase Snackbar.

Entonces voy a colocar Snackbar y es muy similar su utilidad como el anterior, los Toast.

Vamos a hacer aquí, .make y lo primero que nos está pidiendo es un view, un view que voy a poder estar recolectando en mi método onClick, este que está aquí, este objeto v, puedo colocarlo aquí, v, lo siguiente que debemos mostrar aquí es el texto que se va a mostrar en el Snackbar.

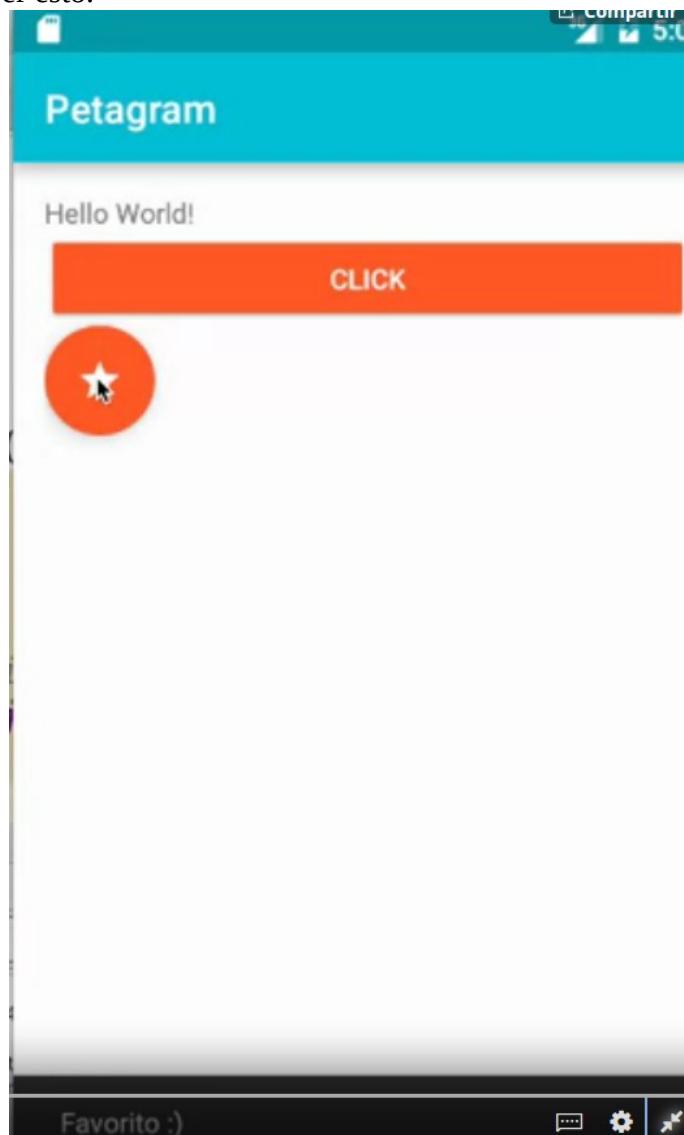
Puedo obtener el texto como lo hice en el ejemplo anterior, getResources.

Es s R.string porque es un string, un recurso de tipo string al que voy a accesar a .mensaje.

y finalmente colocamos la duración de nuestro Snackbar. Para esto pues, como observas es similar a lo que tenemos acá en el Toast pero lo haré a través de la clase Snackbar. Y lo vamos a decir que también sea un length short,

```
id onClick(View v) {  
    st.makeText(getApplicationContext(), getResources().getString(R.string.mensaje), Toast.LENGTH_SHORT);  
    bar.make(v, getResources().getString(R.string.mensaje), Snackbar.LENGTH_SHORT).show();  
}
```

Esto es todo para hacer un Snackbar sencillo, muy simple y entonces pues vamos a guardar nuestro archivo y vamos a correr esto.

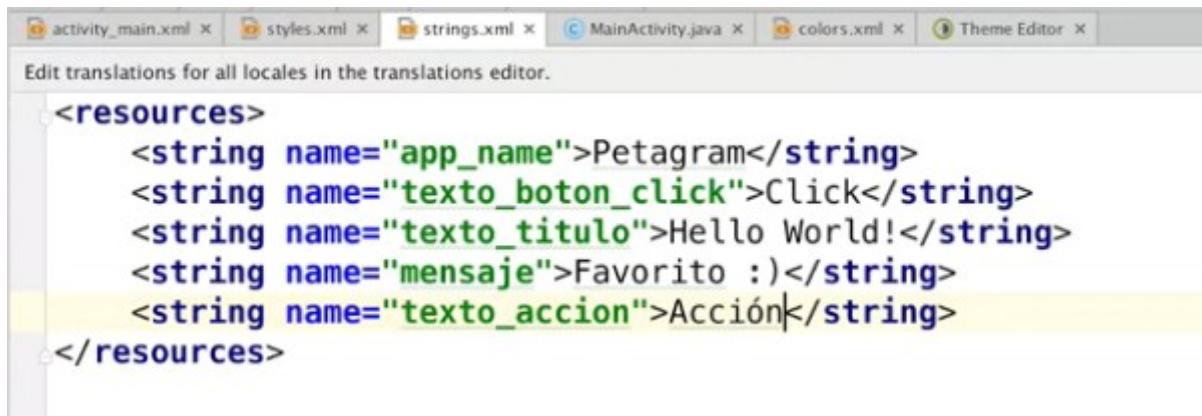


Snackbar son los que han venido a reemplazar a los Toast.

Este Snackbar lo puedo personalizar mucho más, puedo añadirle más cosas y puedo añadirle por ejemplo pues que tenga a lo mejor una acción.

Puedo decirle que cuando aparezca aquí favorito, tenga de este lado un botón o un texto que funcione como un botón que sea una acción, okey, que sea algo a realizar.

Entonces vamos primeramente a nuestro archivo strings a definir el texto que va a estar aquí y voy a poner acá strings.



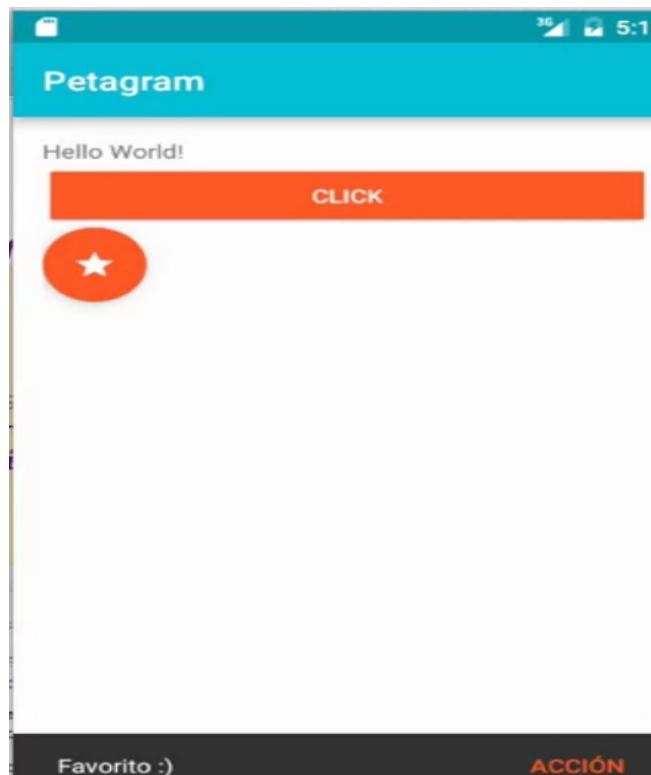
Vamos a poner texto de acción, y luego voy a regresar a mi `MainActivity.java` donde estoy trabajando y tengo aquí mi `Snackbar` donde está el `.show`.

Entonces lo primero que voy a colocar es la instrucción setAction. SetAction y lo primero que recibe es el nombre o el texto que va a aparecer como el texto indicador de la acción. En este caso el texto pues lo tenemos, en nuestro archivo string, getString R.string, se llama texto accion, texto accion. Y este va a recibir un listener también un listener que va a estar pues atento, que va a estar escuchando cuando le demos clic a ese texto que dice acción.



Entonces yo puedo decir que cuando alguien le de clic, bueno en este caso que utilice el log y que nos muestre un mensaje. Yo lo puedo poner snackbar, y le puedo decir click en Snackbar, click en Snackbar y finalmente .show.

Ésto nos da:



Al presionar el botón, lo que dice log, va a verse reflejado en el logcat

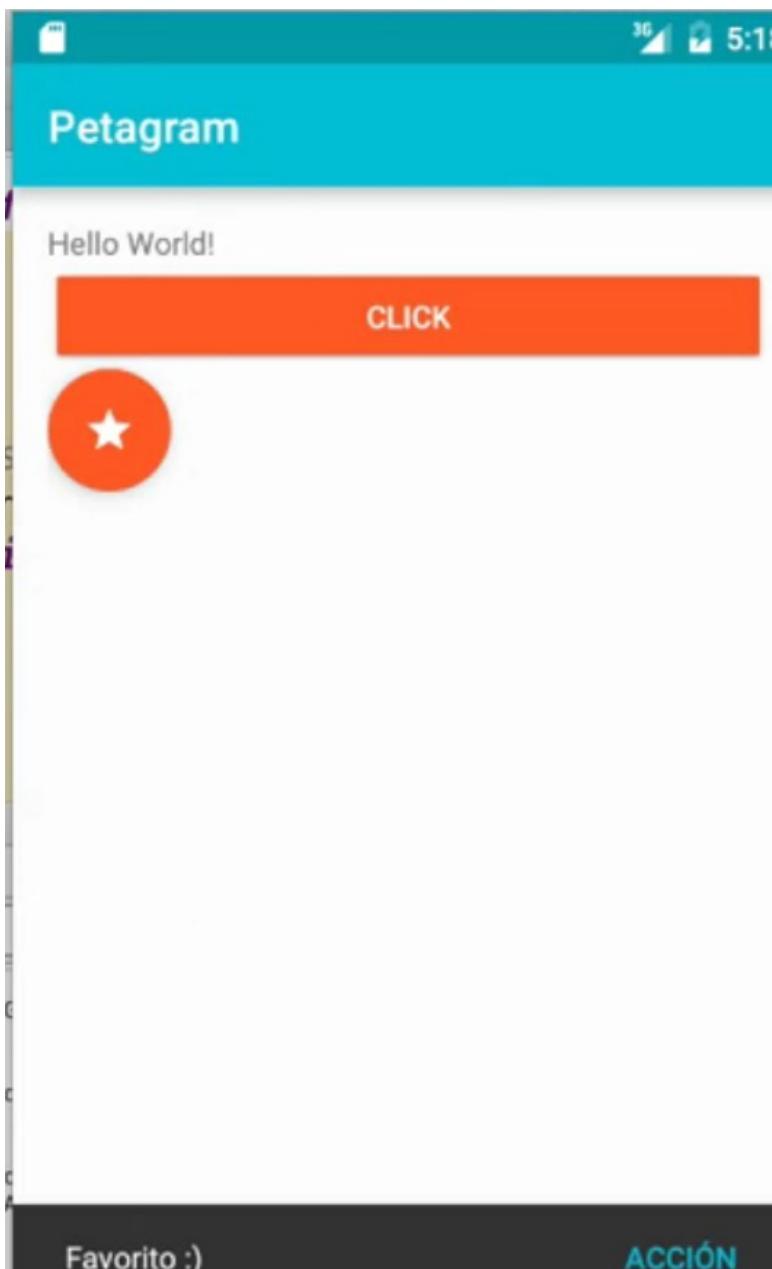
A screenshot of the Android Logcat tool. The log shows several entries, with the last few being:

```
03-11 17:16:26.574 1301-1320/? I/ActivityManager: Displayed com.anncode.petagram/.MainActivity: +1s701ms
03-11 17:16:30.578 3586-3586/? I/SNACKBAR: Click en Snackbar
03-11 17:16:35.783 1840-3604/? V/ConfigFetchTask: ConfigFetchTask getDeviceDataVersionInfo(): ABFEt1WAub4L4G3p-VS1JGMuVzjvZ3k1MxYJADNQXqFDa1vsN0qNRK0GT87MoESHJryjX_
03-11 17:16:35.798 1840-3604/? I/GoogleelBRIConnFactory: Usina nlaform SSLCertificateSocketFactory
```

Podemos personalizar esto de una forma queno sé, si quisieramos cambiar el color del Snackbar, pues simplemente lopodemos hacercon la instrucción después de setAction o antes de setAction cómo gustes; con setActionTextColor podemos decir de qué color queremos que se vea por acá.

En este momento está tomando el color de accent, pero si te gustaría tomar otro, podemos definir aquí que getResources.getColor y vamos a darle R.color. a lo mejor le decimos que queremos el color primario, un color primario que habíamos definido.

```
Toast.makeText(getApplicationContext(), getResources().getString(R.string.mensaje), Toast.LENGTH_SHORT)
        .setAction(getResources().getString(R.string.texto_accion), new View.OnClickListener()
    {
        @Override
        public void onClick(View v) {
            Log.i("SNACKBAR", "Click en Snackbar");
        }
    })
.setActionTextColor(getResources().getColor(R.color.colorPrimary));
.show();
```



Bien, entonces otra cosa que podemos hacer en el Snackbar es que cuando está activo, pues podemos, podríamos eliminarlo deslizándolo.

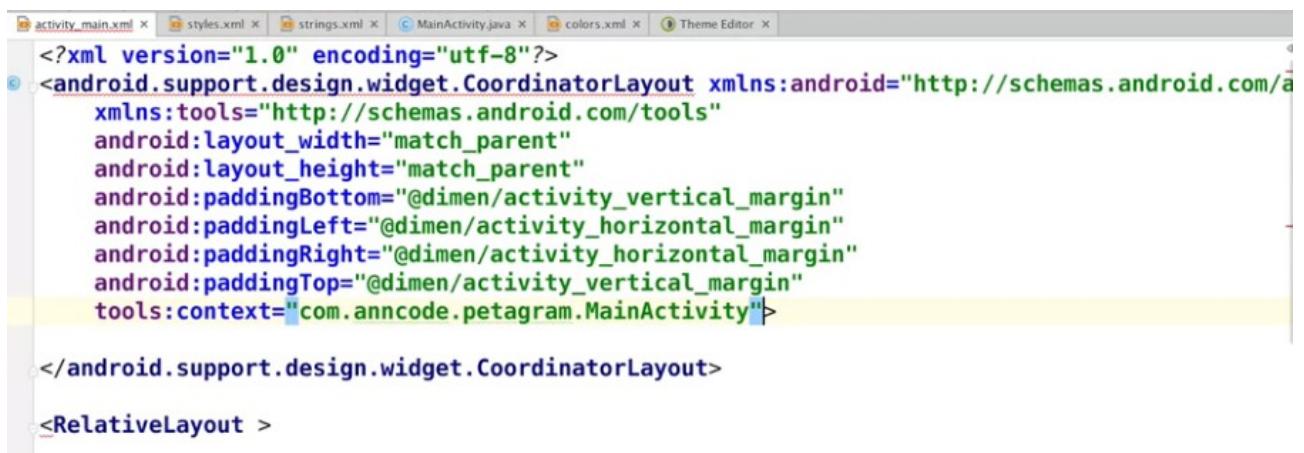
Entonces vamos a colocar un elemento adicional para que podamos aquí nosotros simplemente dando un swipe, quitar el Snackbar. Y entonces esto no está funcionando de esa forma porque el

Snackbar tiene que estar corriendo sobre un elemento, un layout que se llama layout coordinador o coordinator layout.

Entonces, simplemente vamos a ir a nuestro archivo activity main, ahí nuestro activity main.xml y vamos a definir una etiqueta coordinator layout.

Esta también está en la librería de diseño, entonces vamos a escribir antes de relative layout, android.support.design.widget y coordinator layout.

Aquí está, voy a cerrar las etiquetas, ahí. Entonces tengo la etiqueta abierta y la etiqueta cerrada. Acá abajo está mi relative layout.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.anncode.petagram.MainActivity">

</android.support.design.widget.CoordinatorLayout>

<RelativeLayout >
```

Lo que voy a hacer a continuación simplemente todos los atributos que tiene new RelativeLayout, todos los vamos a cortar, los vamos a cortar así, y se lo vamos a asignar al coordinator layout.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.anncode.petagram.MainActivity">

<RelativeLayout >

    <TextView
        android:id="@+id/tvTitulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/texto_titulo" />

    <Button
        android:id="@+id/btnMiBoton" />
```

Como observas, pues en relativeLayout nos está apareciendo aquí, el mensaje es porque simplemente necesita sus atributos obligatorios. Okey, vamos a ponerle match parent, en width y en height, match parent y match parent.

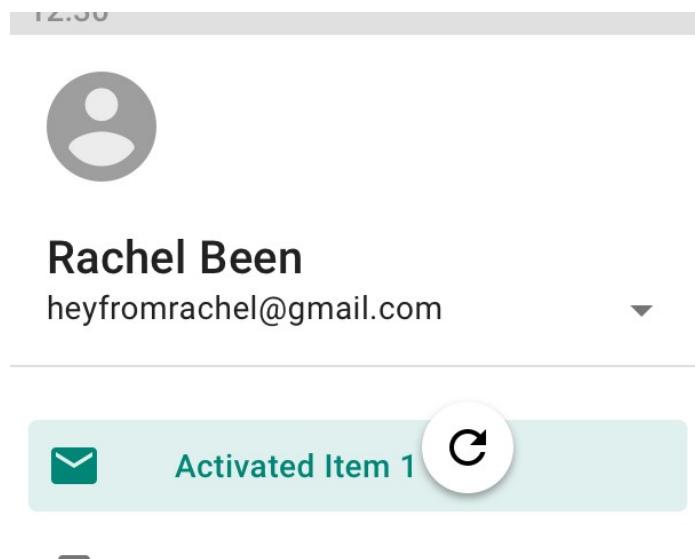
```
<RelativeLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
>
```

En un coordinator layout, es un nuevo elemento de material design, también una nueva etiqueta, vamos a poder hacer este efecto, que cuando nosotros hagamos swipe la notificación se elimine o se quite.

Entonces, y esto va a suceder automáticamente.

Refresh Indicator

El refresh indicator es el que aparece cuando por ejemplo, se hace swipe desde la parte superior de una aplicación hacia abajo para que muestre, por ejemplo, nuevos mensajes en el mail, o nuevos estados en Facebook o nuevas publicaciones en Twitter.



El refresh indicator indica que se está realizando una nueva petición y trabaja sobre una lista de elementos.

Bien, lo primero que vamos a hacer es iremos a nuestra pantalla, iremos aquí a nuestra aplicación y vamos a estar en el modo de texto. En el modo de texto y vamos a colocar arriba de nuestro TextView, vamos a dar un par de saltos de línea, allí vamos a colocar una pequeña lista antes del hello world, antes del hello world que tenemos por aquí, antes de todos los elementos que hemos ido incrustando.

Bien, este elemento lo vamos a encontrar en android.support.v4.widget.swipeRefreshLayout, así lo vamos a conocer swipeRefreshLayout a nivel de código. Entonces en width, vamos a colocar match parent y en height vamos a colocar wrap content.

```
<android.support.v4.widget.SwipeRefreshLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
|  
</android.support.v4.widget.SwipeRefreshLayout>  
  
<TextView  
    android:id="@+id/tvTitulo"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/texto_titulo" />
```

Y entonces se ha añadido una, un sistema de etiquetas donde esta etiqueta nos está indicando de que está lista para recibir una etiqueta hijo, y entonces nuestra etiqueta hijo será nuestro elemento ListView.

```
<android.support.v4.widget.SwipeRefreshLayout  
    android:id="@+id/sfiMiIndicadorRefresh"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
  
<ListView  
    android:id="@+id/lstMiLista"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">  
  
</ListView>  
  
</android.support.v4.widget.SwipeRefreshLayout>
```

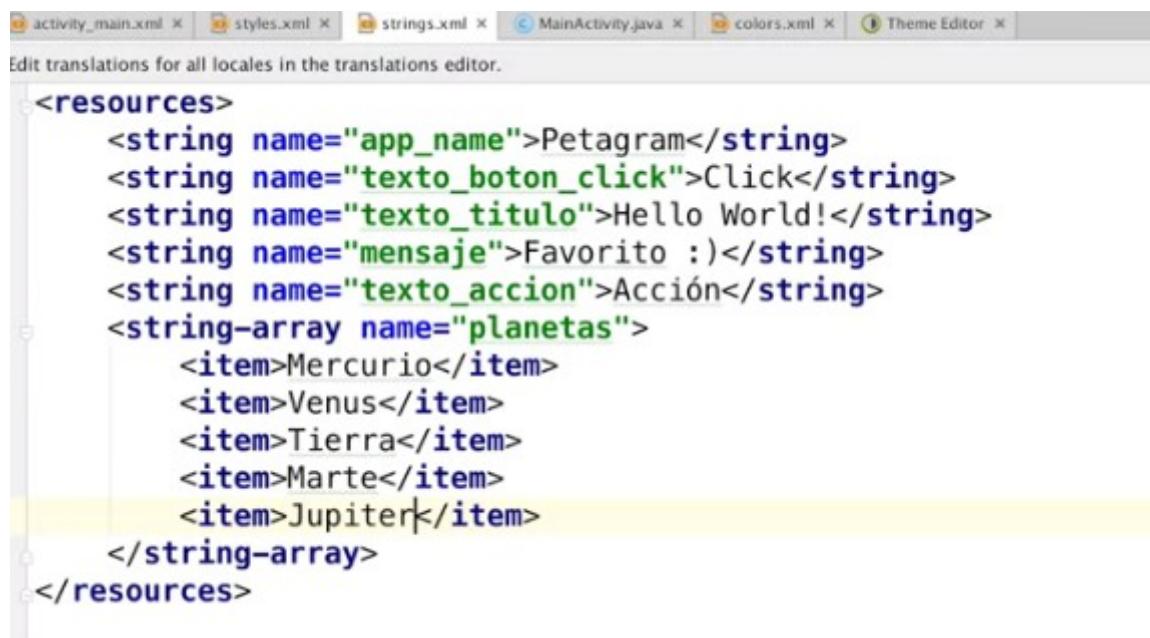
Recordar de poner los ids siempre

Para que el textView no quede encimado, se le indica que va debajo de la lista:

```
<TextView  
    android:id="@+id/tvTitulo"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/texto_titulo"  
    android:layout_below="@+id/sfiMiIndicadorRefresh"  
    />
```

Bien. Ahora, yendo a nuestro archivo strings, vamos a declarar en nuestro archivo strings todos los elementos que se van a mostrar en esta lista; en mi archivo strings voy a declarar una nueva etiqueta que es una etiqueta string array que le vamos a poner nombres.

Vamos a poner una lista de planetas, Mercurio, Venus, Tierra, Marte y Júpiter. Entonces esta lista la vamos a dejar ahí.



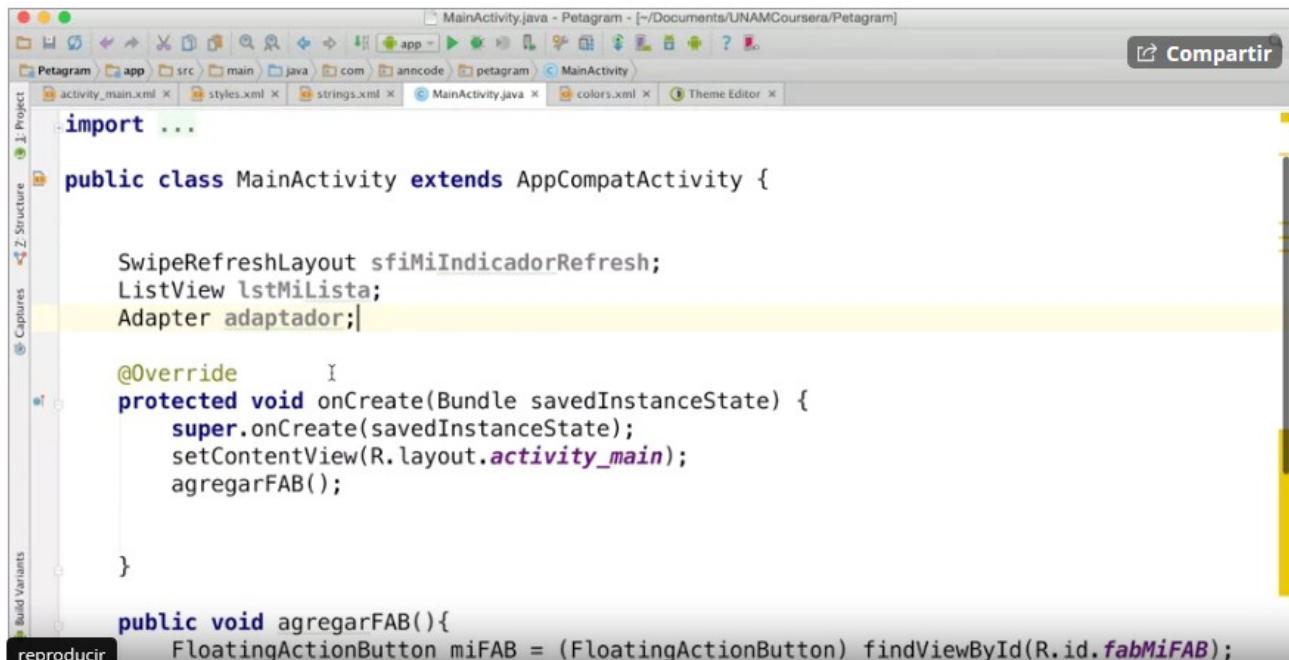
```
<resources>
    <string name="app_name">Petagram</string>
    <string name="texto_boton_click">Click</string>
    <string name="texto_titulo">Hello World!</string>
    <string name="mensaje">Favorito :)</string>
    <string name="texto_accion">Acción</string>
    <string-array name="planetas">
        <item>Mercurio</item>
        <item>Venus</item>
        <item>Tierra</item>
        <item>Marte</item>
        <item>Jupiter</item>
    </string-array>
</resources>
```

Ahora a nuestra clase, a nuestro activity main, nuestro main activity, vamos a ir a nuestro main activity, específicamente en nuestro método on create.

En nuestro método on create debajo de nuestro botón agregarFAB y ahí vamos a empezar a declarar nuestro swipeIndicator.

Entonces voy a, antes de mi método, voy a declararlo como una variable global, swipe, swipeRefreshLayout, le voy a poner conviene colocar el mismo nombre de el id, como los nombres y mis objetos en mi código Java.

Entonces ahí está, voy a colocar también mi ListView arriba, mi ListView, le voy a poner mi lista, sí le pusimos, vamos a revisar nada más, el lst, mi lista y nuestra lista para que se puedan mostrar los nombres o listas de planetas va a requerir de un adaptador, un objeto de tipo adaptador que vamos a colocar un adapter.



The screenshot shows the Android Studio interface with the project 'Petagram' open. The code editor displays the MainActivity.java file. The code defines a class MainActivity that extends AppCompatActivity. It declares variables for SwipeRefreshLayout, ListView, and Adapter. The onCreate method is overridden to set the content view to activity_main and add a floating action button. A public method agregarFAB is also defined.

```
import ...  
  
public class MainActivity extends AppCompatActivity {  
  
    SwipeRefreshLayout sfiMiIndicadorRefresh;  
    ListView lstMiLista;  
    Adapter adaptador;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        agregarFAB();  
  
    }  
  
    public void agregarFAB(){  
        FloatingActionButton miFAB = (FloatingActionButton) findViewById(R.id.fabMiFAB);  
    }  
}
```

Este objeto nos va a servir como auxiliar para estar manejando todos los datos de nuestro archivo strings y pasarlo aquí a mi lista, a mi ListView mi lista.

Ahora aquí en mi método on create, voy a empezar cada uno de estos elementos sfi, voy a indicar con esta sintaxis.

Tal vez esta sintaxis no la hemos explicado detenidamente, pero lo vamos a ver cuando estemos manipulando views.

Básicamente lo que estamos haciendo es tomando nuestro objeto, aquí está nuestro objeto, esta es su definición, y normalmente como estamos acostumbrados es a hacer new y a instanciar el objeto.

En este caso en este objeto no lo necesitamos instanciarlo, no lo necesitamos poner en memoria porque este objeto ya existe, este objeto es un elemento que está en nuestro, declarado en nuestro xml, es un objeto que ya se ha referenciado, ya vive en la memoria.

De hecho nuestro archivo r.java lo que cacha ahí en esa variable es la dirección de memoria donde está viviendo este elemento.

Entonces, este elemento ya existe, no necesito instanciarlo y colocarle new.

Lo que tengo que colocar es la instrucción FindViewById, es decir esta instrucción FindViewById lo que va a hacer es encontrar un view cuyo id sea lo que yo coloque aquí.

Entonces va a encontrar un view, en dónde va a buscar los views, pues en nuestro archivo que está repleto de views, nuestro activity main.xml, nuestro layout, y ahí adentro tenemos que decirle si le estamos diciendo encontrar el view cuyo id sea este.

```
sfiMiIndicadorRefresh = findViewById(R.id.sfiMiIndicadorRefresh)
```

Tenemos que darle el id a través de nuestro archivo R. R.id y el id es el mismo que hemos asignado o que hemos colocado en nuestro archivo xml, sfiMiIndicador Refresh, ese es el id que yo debo estar referenciando. SfiMiIndicadorRefresh.

Bien, si colocamos aquí el punto y coma, observamos qué es lo que está mostrando este warning. Que nos dice que ha encontrado un view, ha encontrado un view pero requiere un elemento que se llama swipeRefreshLayout.

Entonces, cuando este método dice FindViewById, lo que hace es encontrar el view más genérico o el elemento más genérico.

Entonces, en Android podemos ejecutar un casteo, un casteo de este elemento que es un view, a nuestro caso lo que necesitamos un swipeRefreshLayout. Y es el mismo caso cuando trabajamos el Floating Action Button, declaramos nuestro botón, y también aquí ejecutamos nuestro casteo del elemento view que había encontrado.

```
sfiMiIndicadorRefresh = (SwipeRefreshLayout) findViewById(R.id.sfiMiIndicadorRef
```

Entonces, ahora sí, este, este elemento XML que está aquí, ahora sí ya lo puedo manipular a partir de este objeto que he creado con código.

Haré lo mismo con MiLista, lst, lista, haré mi casteo, ListView y le diré findViewById R.id.lstMiLista. Entones, ahí está, lstMiLista.

Como dijimos, para poder llenar esta lista, que finalmente es lo que quiero estar mostrando, voy a necesitar un adaptador.

Un adaptador que voy a colocar aquí en mi objeto lstMiLista, y le voy a poner .setAdapter, setAdapter. Y esto va a recibir new un ArrayAdapter. Una colección de Strings.

```
sfiMiIndicadorRefresh = (SwipeRefreshLayout) findViewById(R.id.sfiMiIndicadorRefresh);
lstMiLista = (ListView) findViewById(R.id.lstMiLista);

String[] planetas = getResources().getStringArray(R.array.planetas);
lstMiLista.setAdapter(new ArrayAdapter(this, android.R.layout.simple_list_item_1));
```

Este ListView, pues, está definido por una serie de items, una serie de elementos.

Para empezar, este ListView, pues, es un ListView sencillo.

No se va a ver el, el sub item que aquí aparece. En realidad, solamente se ven, por ejemplo, el item 1, item 2, item 3, item 4.



Entonces, este ListView está compuesto de tal forma que dentro de sí cada item tiene un TextView.

Tiene un TextView disponible para que podamos rellenarlo, para que podamos colocarle en nuestro caso para cada item la lista o el elemento de los planetas, cada elemento de los planetas.

Entonces si por ejemplo vamos al recurso, solamente tiene definido un TextView.

Entonces solamente es un TextView que nos está ayudando para que cuando le toque colocarle Mercurio, por ejemplo, pues Mercurio se asigne a ese elemento TextView y entonces pueda ser mostrado en la lista. Mercurio, Venus, Tierra y Marte, porque pues bueno el TextView solamente está compuesto por una lista de items.

Un item que está definido por un layout que con el tiempo se está reciclando o se está reutilizando para colocar todos los planetas que necesito. Ese item o ese layout es este elemento separado.

Bien, el último parámetro que nos pide es la lista de planetas.

```
MiLista.setAdapter(new ArrayAdapter(this, android.R.layout.simple_list_item_1, planetas));
```

Y entonces lo que vamos a hacer es que a nuestro swipe, a nuestro swipe indicator que tenemos acá arriba vamos a ponerle un Listener, vamos a ponerle SetOnRefreshListener.

```
sfiMiIndicadorRefresh.setOnRefreshListener();
```

Es decir todos estos Listener, como su nombre lo dice, siempre van a estar escuchando, siempre van a estar atentos.

En este caso para que cuando alguien haga ese gesto de refrescar, de hacer un swipe hacia abajo, entonces SetOnRefreshListener va a recibir new y automáticamente con un control barra espaciadora ya me está sugiriendo que la interfaz onRefreshListener.

Entonces vamos a dar clic ahí y yo no muevo, ningún botón, no muevo nada, automáticamente todo el código es generado.

```
lstMiLista.setAdapter(new ArrayAdapter(this, android.R.layout.simple_list_item_1, p  
sfiMiIndicadorRefresh.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener  
    @Override  
    public void onRefresh() {  
        |  
    }  
});
```

Entonces ahora ya tengo un método que se llama on refresh. On refresh que aparte de que va a estar mostrando nuestra bolita haciendo como un loading, pues además yo puedo definir aquí adentro de este método on refresh qué es lo que va a proceder a realizarse.

Entonces yo puedo colocar aquí un método o puedo hacer que ocurra algo en mi lista.

Entonces yo podría hacer que vamos a poner por aquí un método que podría ser, un método que se llame refrescando contenido.

Este método refrescando contenido es el que vamos a estar llamando aquí en on refresh.

```
sfiMiIndicadorRefresh.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener  
    @Override  
    public void onRefresh() {  
        refrescandoContenido();  
    }  
});  
  
public void refrescandoContenido(){  
    String[] planetas = getResources().getStringArray(R.array.planetas);  
    lstMiLista.setAdapter(new ArrayAdapter(this, android.R.layout.simple_list_item_1, p
```

Para eso pues vamos a colocar de nuevo estas dos líneas donde tenemos nuestros planetas, nuestra lista de planetas y se la volvemos a asignar y ya.

Y entonces una vez que you se haya mostrado pues ahora debemos colocar el indicatorRefresh, set Refreshing, ahora en falso.

```
public void refrescandoContenido(){  
    String[] planetas = getResources().getStringArray(R.array.planetas);  
    lstMiLista.setAdapter(new ArrayAdapter(this, android.R.layout.simple_list_item_1, p  
    sfiMiIndicadorRefresh.setRefreshing(false);  
}
```

Tú en ese momento, bueno, en el método refrescando el contenido podrías decidir que se rellene ahora con otra lista o que a lo mejor consulte un web service y traiga nuevos datos, y traiga los nuevos datos concatenados con los demás.

CardView

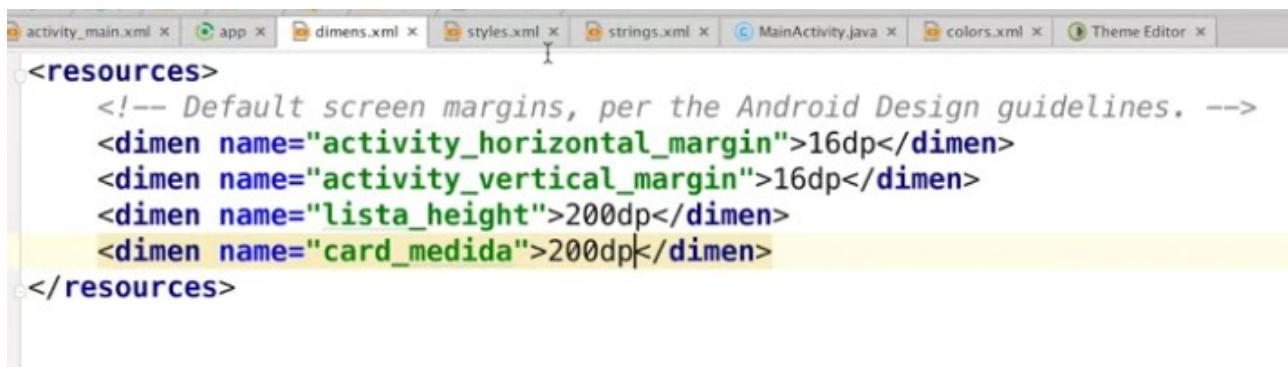
Sirve para presentar listas en tarjetas de forma más amigable.

Entonces vamos a abrir nuestro archivo gradle, lo tengo aquí en modo app.

Aquí está, aquí está abierto mi archivo gradle y aquí en seguida de esta dependencia que acabamos de colocar, también vamos a colocar nuestra dependencia para CardView.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.android.support:design:23.2.1'
    compile 'com.android.support:cardview-v7:23.2.1'
}
```

Y en el archivo dimens:



```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="lista_height">200dp</dimen>
    <dimen name="card_medida">200dp</dimen>
</resources>
```

Y en activitymain:



```
<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_star"
    android:layout_below="@+id/btnMiBoton"
    android:id="@+id/fabMiFAB"
/>

<android.support.v7.widget.CardView
    android:layout_width="@dimen/card_medida"
    android:layout_height="@dimen/card_medida"
    android:layout_below="@+id/fabMiFAB"
>

</android.support.v7.widget.CardView>
```

Con éstas medidas, el CardView queda cuadrado y se ubica por debajo del floating button.

Ahora otra cosa que le vamos a asignar a este card es que, pues normalmente las tarjetas tienen un radio, un borde, un borde redondeado.

Puedes redondearlo conforme a tí te guste más o conforme a la legibilidad que creas correcta, o puedes dejarlo así, pero vamos a colocarle un borde redondeado y para colocarle un borde redondeado voy a necesitar una propiedad que se llama card view.

En vez de colocar android, nuestra propiedad se va a llamar CardView y cómo se observas pues esta propiedad no la tengo dada de alta.

Me pregunta que si quiero traer el namespace de esta etiqueta. Yo podría decirle que sí, quiero que traiga el namespace de la etiqueta y cómo observas, simplemente lo único que hice fué aquí mismo, di con alt enter, cuando me salió la sugerencia, cuando estaba en rojo, alt enter y automáticamente en la parte superior en la etiqueta Padre, me ha traído el namespace de cardview.

The screenshot shows an Android XML layout file in an IDE. The code is as follows:

```
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.anncode.petagram.MainActivity">
```

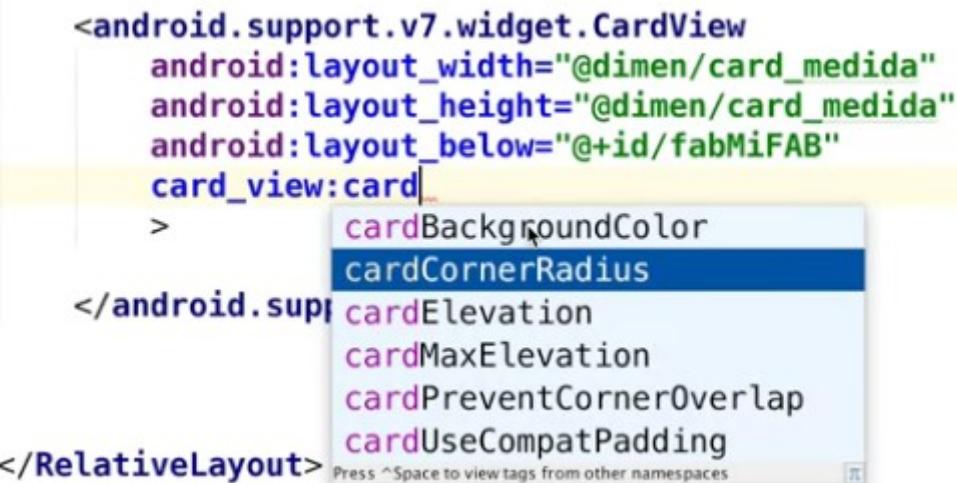
A yellow callout box highlights the line "xmlns:card_view="http://schemas.android.com/apk/res-auto"" with the text "Para que nosotros podamos utilizar estos atributos android dos puntos paddingbottom, paddingleft, necesitamos un namespace que debe estar definido en nuestra etiqueta Padre o en la etiqueta donde estaré utilizando esa propiedad."

`xmlns:android="http://schemas.android.com/apk/res/android"`
`xmlns:tools="http://schemas.android.com/tools"`

Entonces, aquí tengo, este es el namespace para la propiedad android. Acá tengo un namespace para la propiedad tools y si voy a estar utilizando la propiedad cardview, puedo declararla también en la etiqueta Padre y automáticamente esta propiedad es heredada a todos los hijos o sea, está disponible para cualquier etiqueta hijo que quiera utilizarla.

Eh, entonces aquí ya tengo mi etiqueta CardView.

Ya puedo ahora sí, a proceder a colocar CardCornerRadius y entonces con esta propiedad CardView tengo acceso a propiedades que son específicas y exclusivas de CardView.



Entonces voy a colocar un CornerRadius.

Entonces vamos a trabajar el contenido de nuestra tarjeta, con esta etiqueta pues está declarada como una etiqueta Padre, por lo tanto esta etiqueta puede almacenar hijos o puede tener dentro etiquetas hijo.

Entonces voy a declarar ahí dentro, un relativelayout, que es lo que venimos acá manejando, un relativelayout y voy a colocar en width matchparent. Match parent y en height también match parent.

Ahora sí, esto es importante. Cada tarjeta debe tener definido un layout que lo vamos a ver en la siguiente sección. Todos los tipos de layouts que tenemos y para qué sirven.

Hasta ahora hemos estado viendo relativelayout, relativelayout, pero más adelante vamos a estar viendo todos los que tenemos.

Entonces, este layout tiene esto y aquí puedo empezar a definir mis elementos.

¿Cómo quiero que se vea mi tarjeta? Bueno, yo tengo ya lista por acá, una imagen en mi carpeta drawable. Es una imagen que acabo de importar en todas sus densidades, es una imagen que se llama material lollipop. material lollipop esta es mi imagen y básicamente lo que va a mostrar mi cardview va a ser esta imagen y voy a colocar un título de la tarjeta debajo de la imagen y consecuentemente una pequeña descripción de la imagen.

Entonces, cierro eso y dicho lo anterior vamos a empezar a definir nuestra tarjeta de esa forma.

Aquí debe estar la imagen, consecuentemente un título y después una descripción.

Voy a colocar la etiqueta.

Y match view, y match view donde width le vamos a poner match parent, y en height le vamos a poner, ahorita vamos a colocar wrap content y después vamos a ir escalando a ver como de cuánto debe quedar nuestra imagen.

```
<android.support.v7.widget.CardView  
    android:layout_width="@dimen/card_medida"  
    android:layout_height="@dimen/card_medida"  
    android:layout_below="@+id/fabMiFAB"  
    card_view:cardCornerRadius="4dp"  
>  
  
<RelativeLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <ImageView  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:src="@drawable/material_lollipop"  
        android:scaleType="centerCrop"  
    />  
  
</RelativeLayout>
```

Android, 2 puntos, scale type. Scale Type y le voy a poner CenterCrop, CenterCrop.

Entonces lo que hace esta propiedad es que si la imagen de pronto no cabía en esa área, en esa distancia, ahora lo que hace es hacer como un zoom a la imagen y entonces centrarla al full de la imagen para que quepa perfectamente en ese espacio, y la imagen no se pixelea ni se redimensiona no le pasa nada.

Simplemente se entra la imagen para que quepa en el área que estamos definiendo.

Ahora dijimos que debajo de esta propiedad, de esta etiqueta vamos a colocar un text view, es decir un título.

Y para eso, por eso debemos colocar un text view. Le voy a colocar width wrap content y wrap content.

También no olvidemos todos los ids, todos los ids.

Para este lado vamos a poner tvTítulo y le vamos a poner tvTítuloTarjeta.

Los ids no pueden repetirse. Recuerdas que ya teníamos por ahí un text view, que se llamaba tvTítulo. También a la imagen le vamos a poner su id como buena práctica, un img, le vamos a poner imagen.

Vamos a esperar ahí y ahí está.

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/imgImagen"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/material_lollipop"
        android:scaleType="centerCrop"
    />

    <TextView
        android:id="@+id/tvTituloTarjeta"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Ahora el texto que vamos a colocar en esta tarjeta, primero el texto le voy a colocar lollipop y la descripción, vamos a colocar una breve descripción, pues sobre lollipop.

Entonces vamos a nuestro archivo string a declarar un par de strings, un par de variables, string le voy a poner título tarjeta, ahí está.

Le vamos a poner Android 5.0, Android 5.0. Y entonces vamos a poner aquí descripción tarjeta y aquí vamos a poner, esta es la versión en la que se incluyó Material Design.



The screenshot shows the Android Studio interface with the 'strings.xml' file open in the resources editor. The window title bar includes tabs for 'activity_main.xml', 'app', 'dimens.xml', 'styles.xml', 'strings.xml', 'MainActivity.java', and 'Theme Editor'. Below the tabs, there's a message: 'Edit translations for all locales in the translations editor.' On the right side of the window, there are buttons for 'Open editor' and 'Hide notification'. The code in 'strings.xml' is as follows:

```
<sources>
    <string name="app_name">Petagram</string>
    <string name="texto_boton_click">Click</string>
    <string name="texto_titulo">Hello World!</string>
    <string name="mensaje">Favorito :)</string>
    <string name="texto_accion">Acción</string>
    <string-array name="planetas">
        <item>Mercurio</item>
        <item>Venus</item>
        <item>Tierra</item>
        <item>Marte</item>
        <item>Jupiter</item>
    </string-array>
    <string name="titulo_tarjeta">Android 5.0</string>
    <string name="descripcion_tarjeta">Esta es la versión en la que se incluyó Material Design</string>
</sources>
```

Android Lollipop.

Vamos a nuestro main activity y ya podemos añadir la propiedad text primeramente para mi título, para mi TextView título, voy a colocar arroba string título tarjeta, ahí está. Le voy a decir que este TextView debe estar debajo de mi ImageView.

```
<TextView  
    android:id="@+id/tvTituloTarjeta"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/titulo_tarjeta"  
    android:layout_below="@id/imgImagen"  
/>  
  
<TextView  
    android:id="@+id/tvDescripcionTarjeta"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/descripcion_tarjeta"  
    android:layout_below="@+id/tvTituloTarjeta"  
/>
```

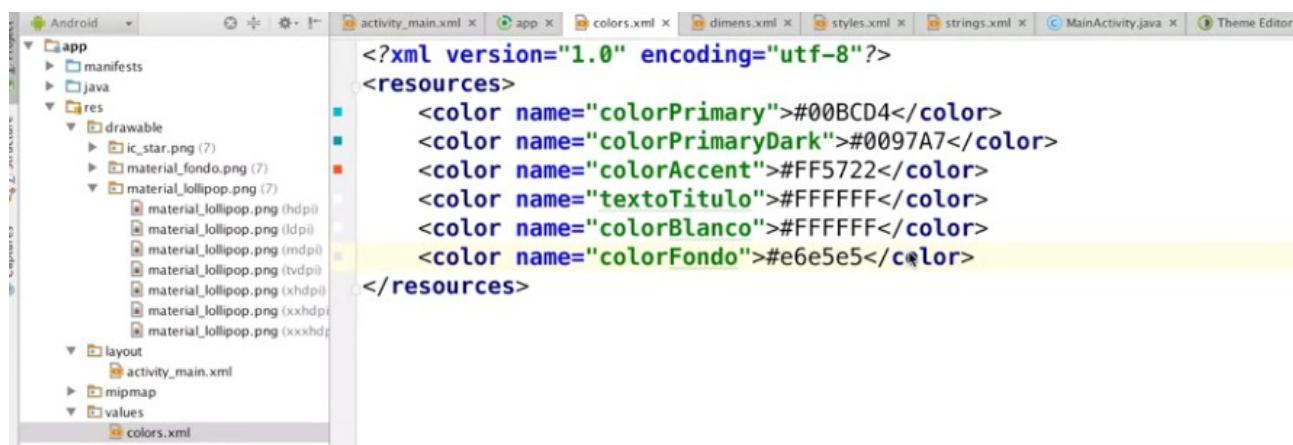
También puede fijarse un estilo para el texto:

```
        android:textStyle="bold"  
        android:textSize="@dimen/card_titulo"
```

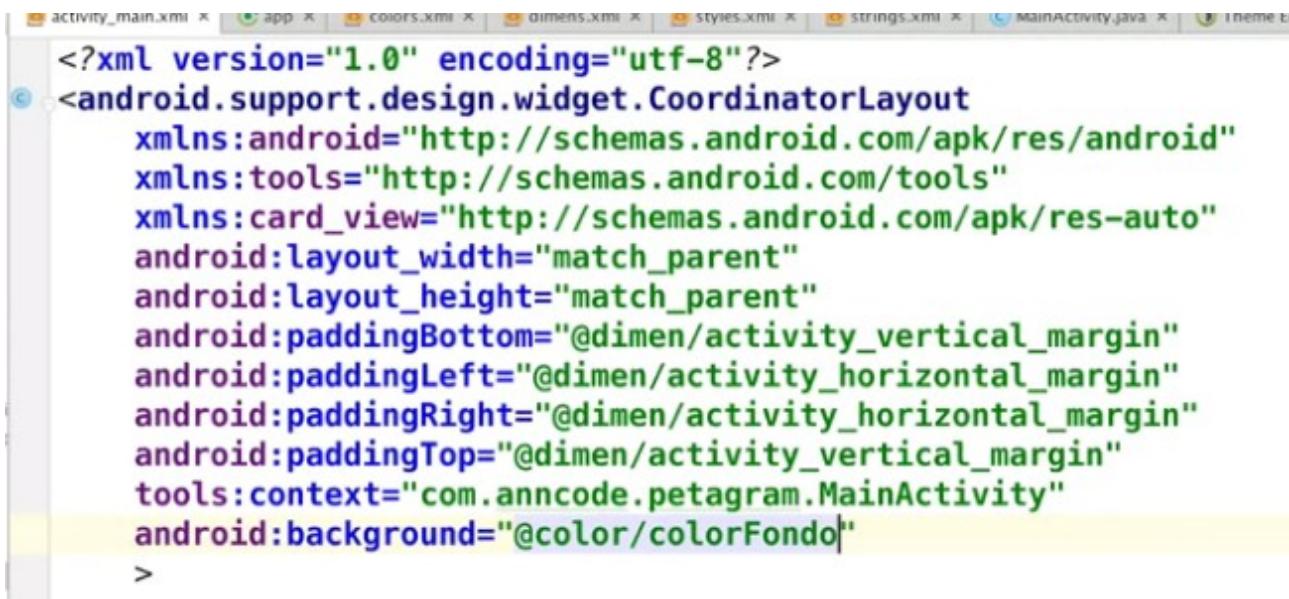
card_título se define en el archivo dimens y va en dp.

Por último, se puede cambiar el tono del color de fondo.

Para eso se va primero al archivo colors y se define uno nuevo:



Y en activitymain:



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.anncode.petagram.MainActivity"
    android:background="@color/colorFondo">
```

Tipos de layouts

Qué son los layouts

Mira todos los views que acabamos de crear.

¿Estás seguro que ese es el orden en que quieres mirarlos? Seguramente no.

Seguramente quieras darles algún otro orden o acomodar uno primero, o que tengan simplemente otra forma.

Bueno, precisamente los layouts nos van a ayudar para eso.

Los layouts es una forma de agrupar todos nuestros views. Todos los views que acabos de hacer podemos darles diferentes orientaciones.

O podemos decir que un elemento siempre esté acomodado a un lado de otro.

O que todos los elementos siempre estén uno consecutivo a otro, tanto vertical como horizontalmente.

Bueno, los layouts nos ayudan a eso.

Los layouts nos van a ayudar a poder dar posición a todos los elementos que están contenidos en él. A todos nuestros views.

Layouts en android

Todos los views que acabamos de ver se están ubicando en nuestra pantalla principal, en nuestra interfaz de usuario. Hasta el momento estos views parecen no tener una estructura de interfaz lógica, es decir, no parecen estar ordenados coherentemente y con la fluidez que necesitan.

Los Layouts nos ayudarán a dar el orden que nuestros view's necesitan. Existen diferentes formas de ordenamiento, es decir, diferentes tipos de Layouts en Android.

Conforme ha pasado el tiempo han ido y venido diferentes tipos de Layouts, te mostramos los oficiales que se encuentran actualmente en la documentación oficial de Android, y te hablaremos de otros que podrías topar con ellos en algún momento.

Los que hay son:

- Linear Layout
- Relative Layout
- List View
- Grid View
- Frame Layout

Lineal layout

En este Layout los elementos son acomodados linealmente, siempre uno en seguida de otro.

Puede ser de forma horizontal o vertical.



En código XML se visualiza de la siguiente forma:

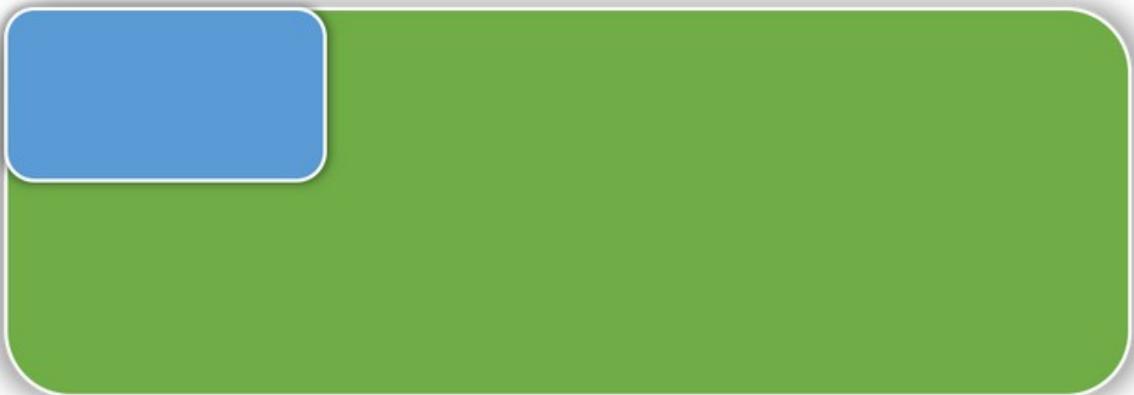
```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical" >  
|  
</LinearLayout>
```

Relative Layout

Este Layout nos ayudará a acomodar nuestros views de forma relativa, es decir la posición de un view siempre será relativa a la posición de otro.

Podemos acomodar los elementos relativamente en los siguientes casos:

- Un view con respecto a su etiqueta padre



```
    android:layout_alignParentLeft="true"  
    android:layout_alignParentTop="true"
```

- Un view con respecto a otro view. Este tipo de Layout se utiliza cuando deseamos crear vistas un poco más complejas.



```
        android:layout_below="@+id/view1"
```

List View

Este aunque podemos seguir usandolo como view dentro de otro layout, ahora se ha convertido en un Layout, el cuál hará que los elementos hijos se acomoden en forma de lista automáticamente scrolleable, este Layout utilizará un el patrón adaptador para colocar los elementos renglón a renglón, dentro de la lista.

Para que este view funcione como un layout, la clase que controla el layout en vez de heredar de la clase Activity deberá hacerlo de ListActivity como se muestra a continuación:

```
public class ListViewActivity extends ListActivity {  
    @Override  
    public void onCreate(Bundle savedInstanceState, PersistableBundle persistentState) {  
        super.onCreate(savedInstanceState, persistentState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

El archivo layout se verá de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<ListView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</ListView>
```

Grid View

Este Layout nos ayudará a ver todos nuestros elementos en forma de grid, algo así como una tabla. De la misma forma que List View, este layout también utiliza un adaptador ListAdapter para insertar cada elemento dentro del grid.

A nivel de código nuestra clase que controla el layout, seguirá heredando de Activity o similar, y nuestro GridView lo llamaremos dentro del método onCreate de la siguiente forma:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v,
                int position, long id) {
            Toast.makeText(HelloGridView.this, "" + position,
                    Toast.LENGTH_SHORT).show();
        }
    });
}
```

Tu adaptador ImageAdapter es una clase que debes crear la cual puede heredar de BaseAdapter como se muestra a continuación:

Estos primeros métodos nos ayudarán a tener el control de la lista de elementos que estarán en el grid, nos ayudarán a obtener datos claves como la cantidad de elementos que existen en la lista, la posición de un elemento en particular, y obtener el item a partir de su id.

```
public class ImageAdapter extends BaseAdapter {  
    private Context mContext;  
  
    public ImageAdapter(Context c) {  
        mContext = c;  
    }  
  
    public int getCount() {  
        return mThumbIds.length;  
    }  
  
    public Object getItem(int position) {  
        return null;  
    }  
  
    public long getItemId(int position) {  
        return 0;  
    }  
}
```

```
// create a new ImageView for each item referenced by the Adapter  
public View getView(int position, View convertView, ViewGroup parent) {  
    ImageView imageView;  
    if (convertView == null) {  
        // if it's not recycled, initialize some attributes  
        imageView = new ImageView(mContext);  
        imageView.setLayoutParams(new GridView.LayoutParams(85, 85));  
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);  
        imageView.setPadding(8, 8, 8, 8);  
    } else {  
        imageView = (ImageView) convertView;  
    }  
  
    imageView.setImageResource(mThumbIds[position]);  
    return imageView;  
}  
  
// references to our images  
private Integer[] mThumbIds = {  
    R.drawable.sample_2, R.drawable.sample_3,  
    R.drawable.sample_4, R.drawable.sample_5,  
    R.drawable.sample_6, R.drawable.sample_7,  
    R.drawable.sample_0, R.drawable.sample_1,  
    R.drawable.sample_2, R.drawable.sample_3,  
    R.drawable.sample_4, R.drawable.sample_5,  
    R.drawable.sample_6, R.drawable.sample_7,  
    R.drawable.sample_0, R.drawable.sample_1,  
    R.drawable.sample_2, R.drawable.sample_3
```

Los últimos dos métodos nos ayudan a setear los datos en cada elemento del grid, y a generar un array de datos dummy.

Y en el layout tendrás algo así:

Y en el layout tendrás algo así:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

Frame Layout

Este Layout es muy común verlo implementado cuando creamos interfaces de Material Design, puesto que en este los elementos siempre están acomodados uno encima de otro, como si fueran capas, justo lo que Material Design nos propone como una forma de ver nuestras vistas, en los ejes x,y,z.

En este layout todos los elementos estarán uno sobre otro, y para lograr el efecto deseado, únicamente debemos darle posición y elevación a nuestros views dentro del layout.

Ver código en <https://www.coursera.org/learn/desarrollo-de-aplicaciones/supplement/EnW2H/frame-layout>

Ciclo de vida de un activity

Ciclo de vida de un activity

Hemos creado una interfaz con algunos Views de material design.

Pero ahora, ¿Que pasaría con tu aplicación o con tu activity cuando de pronto aprietas el botón de back o cuando aprietas el botón de home?. O si de pronto entra una llamada.

¿Que está sucediendo con tu aplicación y con todos tus views? ¿Qué pasa cuando de pronto apagas el teléfono con el botón de encendido? ¿Qué pasa con tu aplicación?

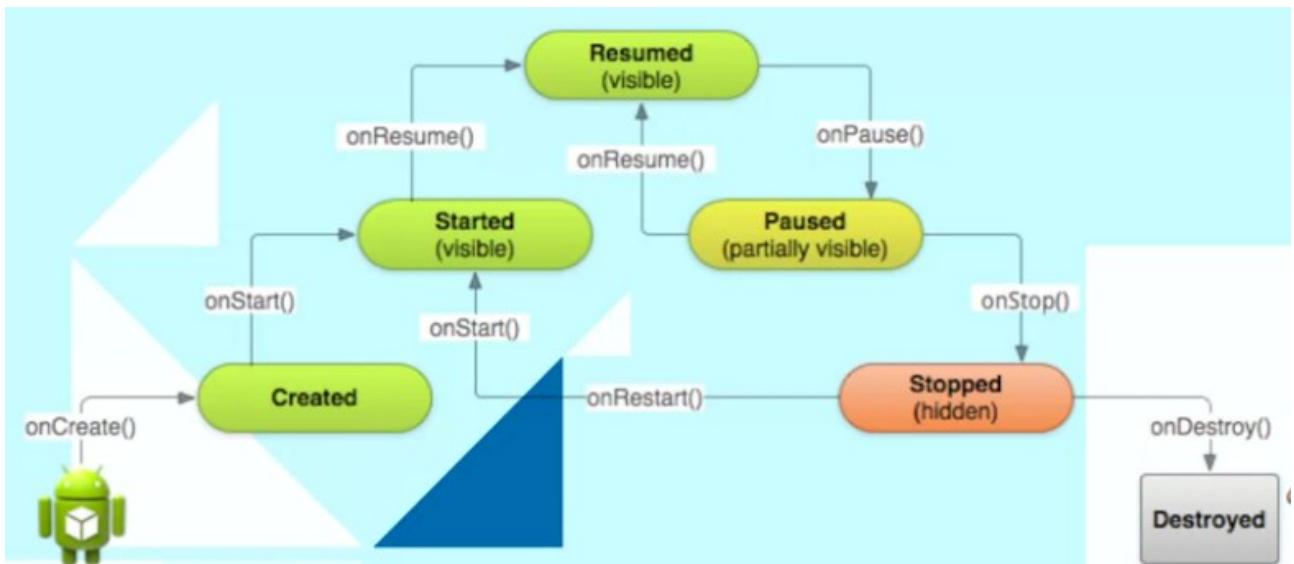
Vamos a ver como controlar el ciclo de vida de un activity.

Y déjame decirte que es muy importante porque precisamente podremos controlar aquellos eventos que se dan cuando entra una llamada o cuando apretamos el botón de encendido.

Y de repente el teléfono se apaga. ¿Qué pasa cuando aprieto el botón de back? Etcétera.

Todo eso lo vamos a ver y vamos aprender a controlar el ciclo de vida de un activity.

Como puedes ver en este gráfico, estamos teniendo el ciclo de vida de una actividad.



- **onCreate:** El ciclo de vida de una actividad siempre va a comenzar con el método On Create. Una actividad siempre va a ser creada. Siempre que abramos una actividad el primer método que se va a ejecutar es el método On Create.
- **onStart:** Posteriormente, la actividad pasará a un estado de On Start.
- **onResume:** Consecutivamente, como estamos viendo aquí, pasaremos a un estado de On Resume. Y por último, la actividad se encontrará en un estado Running, es decir, la actividad está corriendo. Ya todo lo que ves en tu pantalla en ese momento estamos en esa fase, Activity Running.
- **onPause:** El siguiente método que puede ocurrir es el método On Pause. Este método va a ocurrir cuando de repente presiones el botón de Back o de repente presiones el botón de Home.
- **onStop:** El método consecutivo es el método On Stop. Este método también se va a ejecutar cuando presionemos el botón de Back o el botón de Home.
- **onDestroy:** Por último, el método On Destroy. Este método va a ocurrir siempre que una actividad sea destruida. ¿Recuerdas cuando practicamos sobre el Stack de actividades, todas las actividades que se van almacenando en una pila? Bueno, cuando nosotros presionamos el botón de Back la actividad va a pasar del estado Running hasta el estado Destroy, es decir, va a ser destruida.
- **onRestart**

Pero cuando nosotros presionamos el botón de Home en nuestro teléfono la actividad va a pasar de un estado Running a un estado On Stop. Es decir, la actividad no está siendo destruida, simplemente se está manteniendo no visible, en un estado de background. Si en un momento nosotros quisiéramos reanudar la actividad, es decir, si ya no vemos la actividad y presionamos nuevamente el icono para verla. Si la actividad se encontraba en un estado destroy. Es decir que fue destruida, pasará desde el método On Create, On Start y On Resume.

Si nuestra actividad estaba en un estado detenida, On Stop, es decir, presionamos el botón de Home. Nuestra actividad irá al método On Restart. Se iniciará con el método On Start, On Resume, y la actividad comenzará nuevamente en un estado de Running.

Como puedes observar son diferentes eventos que se dan dependiendo de que botón estás presionando o dependiendo de que acción estás haciendo cuando una actividad está corriendo.

Es muy importante controlar estos eventos. ¿Por qué? Imagina que de pronto estás escribiendo un archivo o estás consultando un Web Service, o estás haciendo alguna tarea compleja como por ejemplo podría ser encender el Flash en una cámara.

Todos estos eventos es muy importante controlarlos para evitar que existan errores o que ocurran cosas que no estábamos esperando en nuestras aplicaciones. Vamos a ver un ejemplo de como implementar todos estos métodos y poder controlarlos mejor.

Cuando la actividad se encuentra en un estado en la que está destruida, es decir, apretamos el botón de Back y queremos reanudarla. Al reanudarla pasará a un estado On Create, es decir, la actividad volverá a crearse. Posteriormente a un estado On Start, On Resume. Y finalmente la actividad se encontrará corriendo de nuevo.

Ahora, si lo que sucedió con nuestra actividad es que está en un estado detenida, es decir, apretamos el botón de Home. En ese momento, si queremos reanudarla, pasaremos al método On Restart. On Start, On Resume, y nuevamente la actividad se encontrará corriendo. Es decir, en este momento la actividad no será creada de nuevo, sino simplemente será reanudada en el estado en el que la dejamos.

Es muy importante que estemos controlando todos estos métodos y que estemos controlando todos estos estados que podría tener un activity. Esto es porque imagínate que, por ejemplo, estarías creando un archivo en tu actividad, estarías escribiendo un archivo o estarías consultando un Web Service. O a lo mejor estarías activando el Flash de tu cámara o activando la cámara, etcétera. Todo esto es importante controlarlo.

Que pasará con tu actividad o que pasará con tu aplicación cuando alguien apriete el botón de Back. ¿Qué pasará con tu aplicación cuando alguien pulse el botón de Home? E imagínate que de pronto estás consultando un Web Service o estás descargando información. ¿Qué pasará con esa información que se está descargando? ¿O qué pasará si está encendido el Flash y apretamos el botón de Home? El Flash debería apagarse. Vamos a ver un ejemplo para poder controlar todos estos métodos.

Métodos Callbacks

Aquí tenemos el método onCreate que nos va a presentar a los layouts que tenga activitymain



```
activity_main.xml × MainActivity.java ×
package com.anncode.ciclodevidaactivity;

import ...

public class MainActivity extends AppCompatActivity {

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

}
```

Ahora se va a definir onStart que el IDE ya presenta cómo es el código. Recordar que ésto sobrescribe el método onStart que heredó de AppCompatActivity

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

@Override
protected void onStart() {
    super.onStart();
}

@Override
protected void onResume() {
    super.onResume();
}
```

Los métodos onCreate, onStart y onResume se dan cuando la aplicación está corriendo.

Cuando se detenga la aplicación, se van a usar los siguientes métodos según el caso.

```

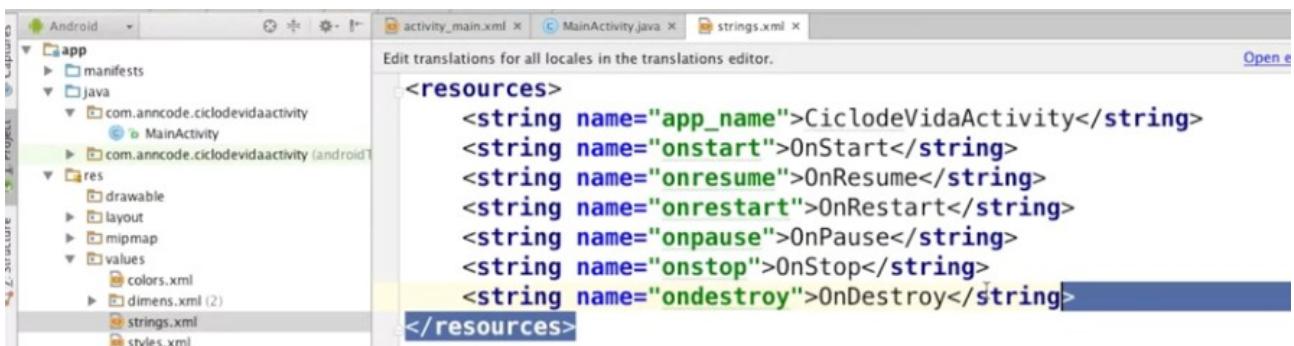
@Override
protected void onPause() {
    super.onPause();
}

@Override
protected void onStop() {
    super.onStop();
}

@Override
protected void onDestroy() {
    super.onDestroy();
}

```

Ahora bien, para que se vea un mensaje en cada una de las situaciones, se va primero al archivo strings para definir los mensajes:



Y luego en el archivo activitymain.java:

```

@Override
protected void onStop() {
    super.onStop();
    Toast.makeText(this, getResources().getString(R.string.onstop), Toast.LENGTH_LONG).show();
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Toast.makeText(this, getResources().getString(R.string.onDestroy), Toast.LENGTH_LONG).show();
}

```

Ojo, se debe cerrar el paréntesis y poner .show() (eso no está porque no entraba en la figura)

Cuando se presiona el botón de back, se ejecuta onPause, onStop y onDestroy. Ésto hace que si, por ejemplo, se estaba creando un formulario, al apretar el botón de back se pierde la información porque se ejecuta el método onDestroy

Si se ejecuta de nuevo, se ejecuta onCreate, onStart, onResume y queda corriendo nuevamente.

Ahora, cuando se presiona el botón de home se ejecuta sólo los métodos onPause y onStop, la actividad no se destruye sino que no queda visible. Ahora, si la reanudamos, se ejecuta onRestart, onStart y onResume; no pasa por onCreate.

Otra cosa que es importante es que cuando se pasa de portrait a landscape (o al revés), hace onPause, onStop, onDestroy, onCreate, onStart y onResume.

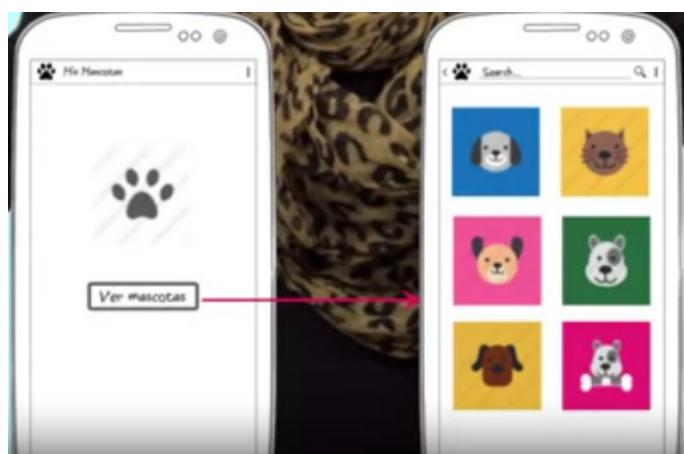
Intents

Iniciando una actividad. Intents

Como se vió anteriormente, una actividad puede iniciar otra tantas veces quiera y quien hace posible ésto es la clase intent.

Los intents son utilizados para la unión entre componentes de una aplicación o incluso componentes que estén fuera de la aplicación.

Funge como pegamento entre una actividad y otra

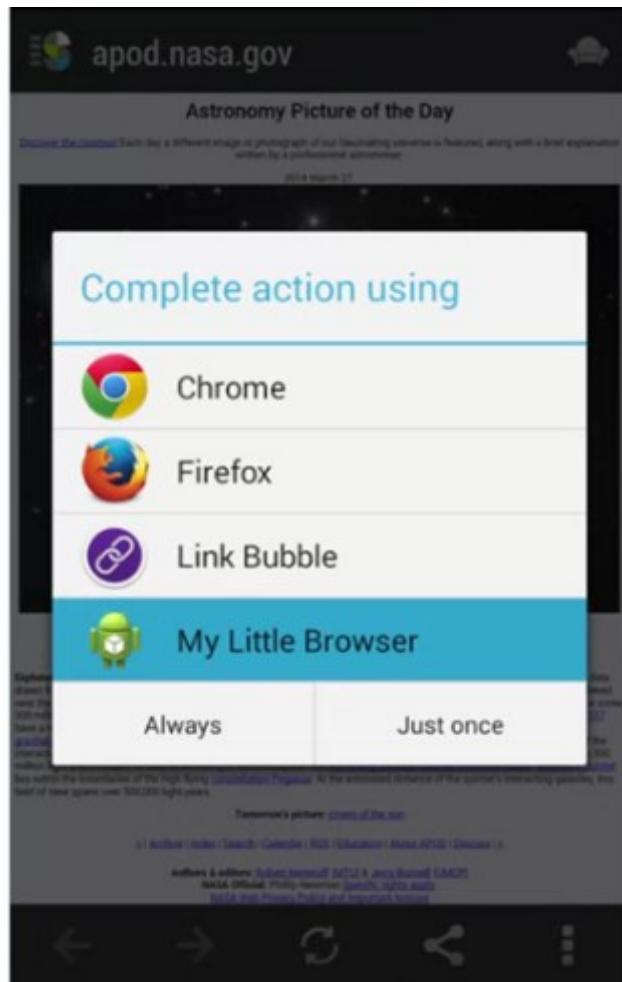


Tipos de intents: explícitos e implícitos

Tenemos dos tipos de intents, los intents implícitos y los intents explícitos.

Los intents implícitos nos van a ayudar a unir componentes de nuestra aplicación con incluso componentes que estén en el sistema o que estén en otras aplicaciones.

Por ejemplo, aquí estamos observando un intent implícito.



Ese intent implícito lo que hace es abrir una dirección web en un navegador o en una aplicación que tenga un navegador como componente, etcétera.

Entonces, los intent implícitos siempre van a ejecutar acciones, acciones que el sistema va a reconocer y de pronto el sistema va a sugerir una aplicación o va a sugerir el navegador web o cualquier otro elemento que pueda abrir este intent implícito.

Por otra parte los intents explícitos son aquellos que nos van a ayudar a unir componentes que estén dentro de una misma aplicación, por ejemplo a unir activities.

Como se está viendo en el código de aquí, tenemos que declarar nuestro objeto intent, intent i, o el nombre que le quieras dar a tu objeto, igual a new intent, y posteriormente colocar dentro del método constructor del intent los siguientes parámetros.

```
Intent i = new Intent(this, DetalleContacto.class);  
startActivity(i);
```

Primero es en dónde estás ubicado, y el segundo parámetro es a dónde quieras ir. Es decir, si estoy en la actividad 1, debo colocar algo así como this de esta actividad coma, quiero ir a la actividad 2.class.

Cuando ya hemos configurado nuestro intent, nuestro objeto intent, simplemente con la siguiente línea de código startActivity, con esta línea disparamos el intent.

Una vez el intent ya esté configurado, startActivity y lo que hace es simplemente ir de la actividad 1 a la actividad 2.

Es probable que también necesites transferir datos de un activity a otro utilizando un intent explícito.

Para hacer esto, utilizaremos un método específico del objeto intent que se llama putExtra.

Como observas aquí en el código, tenemos nuestro objeto intent y estamos utilizando el método .putExtra.

```
String name = "Anahí Salgado";
Intent i = new Intent(this, DetalleContacto.class);
i.putExtra(KEY_EXTRA_NAME, name);
startActivity(i);
```

 Compartir

Este método recibe dos parámetros, el primer parámetro es el nombre del valor y el segundo parámetro es el valor en sí.

Es decir, el método putExtra utiliza el método de clave valor. Tenemos que, debemos tener una forma de reconocer ese valor, y ese valor lo reconoceremos a partir del nombre que definas en el primer parámetro.

Una vez transferido el dato, del otro lado tenemos que cachar ese dato. ¿Cómo lo hacemos?

A partir del objeto bundle. El objeto bundle será un objeto que estará cachando los extras.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detalle_contacto);

    Bundle extras = getIntent().getExtras();
    String name = extras.getString(KEY_EXTRA_NAME);
```

 Compartir

En Android todos los extras son aquellas cosas llamadas, por ejemplo parámetros en web. Cuando estás pasando un parámetro en web, lo puedes hacer a través de una URL.

Aquí en Android todos los parámetros se llaman extras.

Entonces para recibir un extra o un parámetro, lo haremos a través del objeto bundle. Una vez teniendo en este objeto bundle todos los extras o todos los parámetros, podemos ir solicitando uno a uno cada parámetro con el método getString, como lo estamos observando en el código.

Colocaremos extras.getString y lo que llevará a ahí será el nombre del parámetro que habías asignado en el objeto anterior, desde el otro lado de la actividad.

Una vez teniendo getString, esto será almacenado en un objeto de tipo string, si es que así lo solicitaste.

Si estás enviando un número, tendrías que solicitarlo como getInt, si estás enviando un número decimal, tendrías que solicitarlo como getDouble y así sucesivamente dependiendo del tipo de dato que estés mandando, es el mismo tipo de dato que tienes que estar solicitando.

```
Bundle extras = getIntent().getExtras();
String name    = extras.getString(KEY_EXTRA_NAME);
int number     = extras.getInt(KEY_EXTRA_NUMBER);
```

Ejemplo de intents explícitos e implícitos

Vamos hacer un ejemplo donde podamos estar aplicando un item implícito y un item explícito.

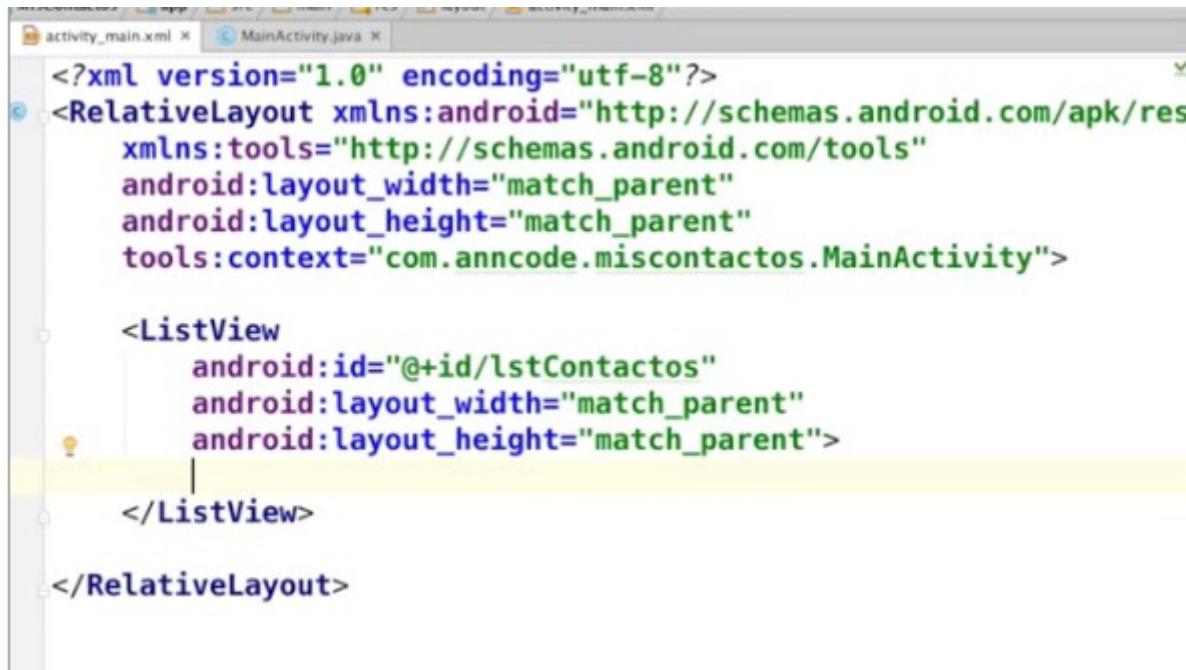
Lo que haré para demostrar todo esto es que vamos hacer una lista de contactos, una de lista de contactos en la que yo pueda tener el nombre de una persona, pueda tener su teléfono y además pueda tener un correo electrónico.

Básicamente lo que hará esta aplicación será que cuando seleccionemos una persona, pasará a otra actividad en donde yo tendré ahí los detalles de esa persona entre ellos su nombre, su teléfono, su correo electrónico.

Al presionar el teléfono, inmediatamente la actividad deberá ejecutar la aplicación de llamadas para ejecutar la llamada. Y también cuando yo presione el botón de email, pues en ese momento se ejecutará un, un chooser o una aplicación que te permitirá elegir entre todas las aplicaciones de correos para enviar un correo a esa persona.

Bien. Vamos abrir, vamos a crear un nuevo proyecto y voy a ponerle Mis contactos y vamos a dar next, vamos a seleccionar una actividad en blanco, vacía, y vamos a esperar a que el proyecto comience a crearse.

Bueno, lo primero que tengo por aquí es mi Layout, y la primera pantalla que voy a mostrar será una lista de contactos.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.anncode.miscontactos.MainActivity">

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

</RelativeLayout>
```

Entonces, voy a quitar este elemento TextView, y voy a sustituirlo por un elemento ListView. Este ListView va ser en su width match_parent y en su height también match_parent, voy a colocar su Id le voy a poner lst contactos. Y estos padding que están por acá me están como estorbando un poco, los voy a quitar.

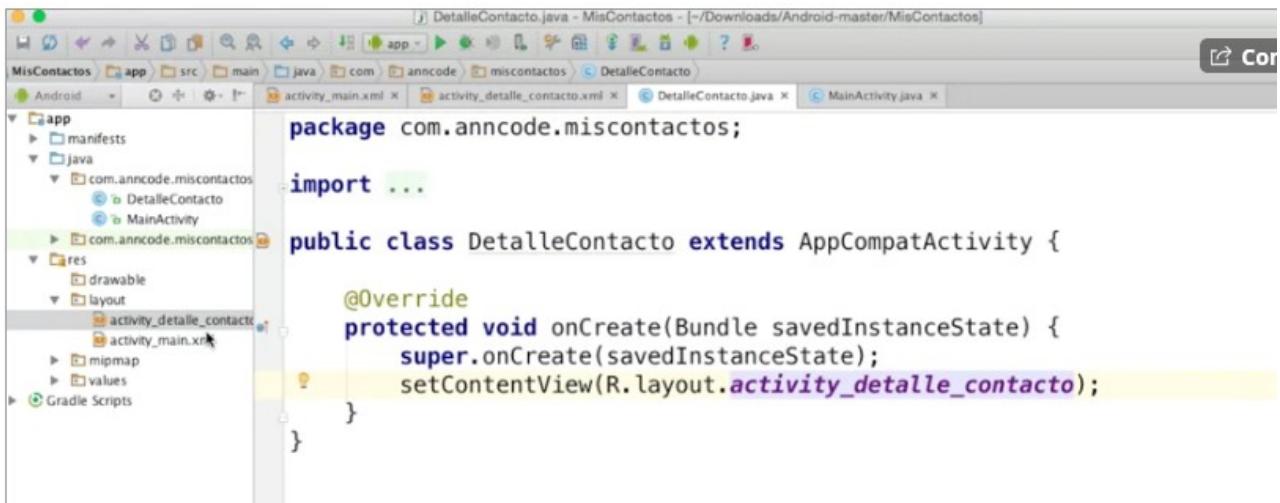
Si observas, estos padding marcan el espacio que hay entre la lista y la pantalla del teléfono, entonces lo que voy a hacer es que lo voy a quitar, así de simple y listo.

Básicamente la lista, vamos a tener aquí una lista de nombres de personas, y que cuando yo le de clic a esa persona debemos pasar a otra actividad, donde ahí se estará mostrando pues el detalle de ese contacto.

Entonces, vamos a hacer nuestro segundo Activity, que va contener el detalle de ese contacto. Voy a darle clic derecho, puedes hacerlo a cualquier nivel del proyecto, y voy a colocar clic derecho new en la opción de Activity, y vamos a seleccionar un Activity Empty, Empty Activity o alguna actividad vacía.

Aquí nos está solicitando el nombre de la segunda actividad, vamos a colocar, detalleContacto, Activity, detalleContacto, así lo vamos a dejar en Layout también, y vamos a dar finish.

Perfecto, ya nos ha creado una segunda actividad que además tiene su Layout por aquí.



```
package com.anncode.miscontactos;

import ...

public class DetalleContacto extends AppCompatActivity {

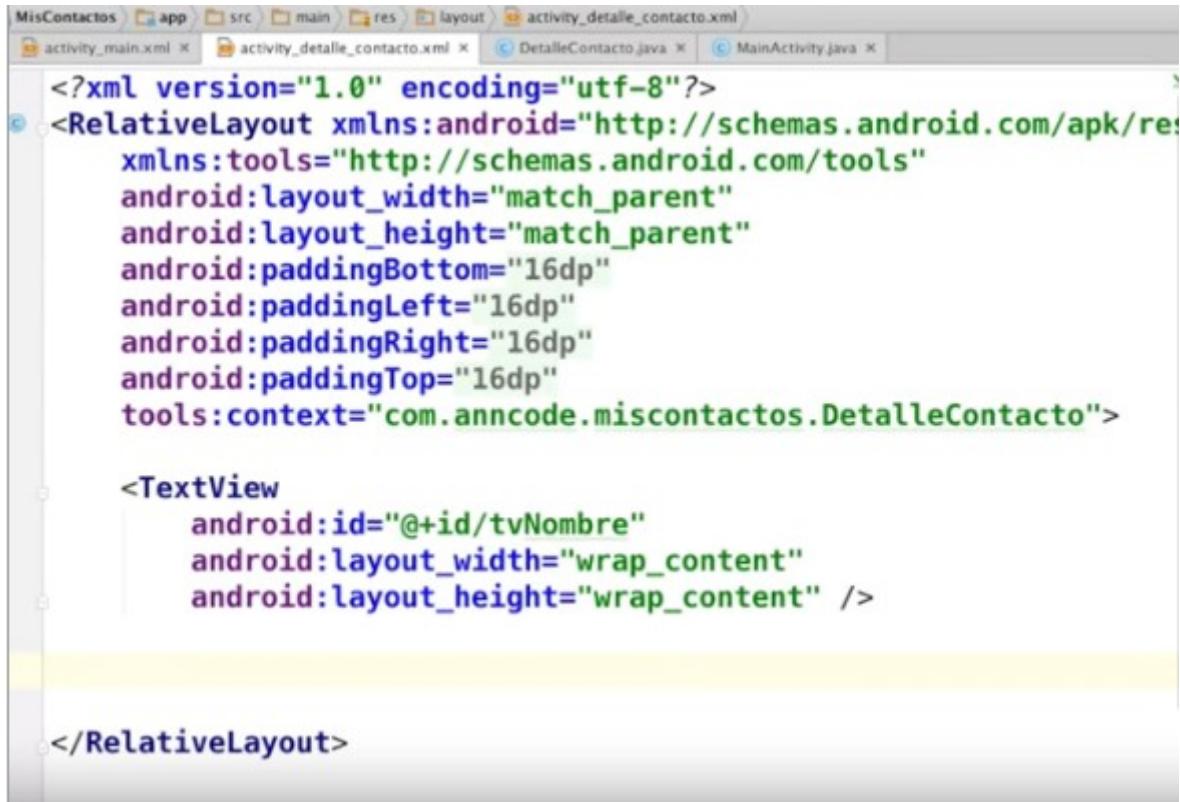
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detalle_contacto);
    }
}
```

Vamos a crear un poquito el detalle, el detalleContacto, vamos a diseñar un poco este Layout cómo se debe ver.

Bueno, lo primero que quiero mostrar en, en grande por aquí, pues es el nombre de la persona por lo tanto necesitaré un elemento llamado TextView.

Lo siguiente, es que quiero mostrar el teléfono de la persona, eso también lo haré en un TextView, y antes del teléfono, antes del teléfono voy a colocar un ícono de un teléfono, entonces estará a un lado.

En el siguiente renglón voy a colocar igualmente ahora un ícono de email, y enseguida el email, entonces por ahí necesitaré algunos recursos.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.anncode.miscontactos.DetalleContacto">

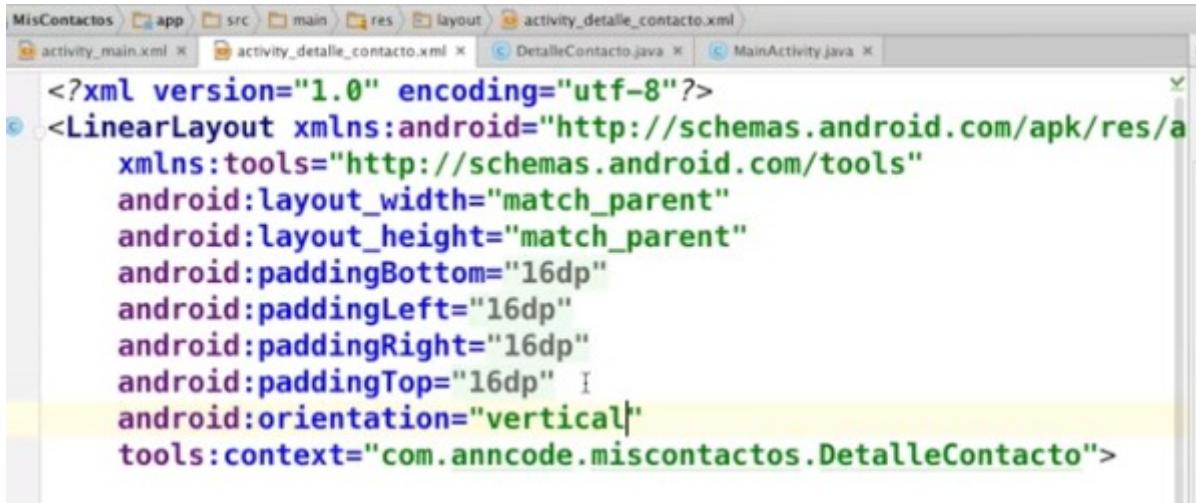
    <TextView
        android:id="@+id/tvNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Vamos a modificar este RelativeLayout, vamos a cambiarlo por un LinearLayout.

Un LinearLayout que sea en posición vertical, el hecho de que yo tenga un LinearLayout en posición vertical, me va permitir que todos mis elementos automáticamente en la forma consecutiva en que están ordenados, van a estar uno debajo de otro.

Entonces voy a colocar aquí en vez de RelativeLayout, LinearLayout, y debo colocar también la orientación de este LinearLayout. Entonces voy a poner android:orientation, y vamos a decir vertical.



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:orientation="vertical"
    tools:context="com.anncode.miscontactos.DetalleContacto">
```

Bien, y hasta aquí mi TextView, hasta este momento no veo nada porque el TextView no tiene ningún texto. Entonces, voy a ir a mi archivo string, para declarar los textos que me, que se, me funcionaran en este caso solamente como muestra para poder estar viendo un poco cómo se estará comportando mi aplicación.

Bien, entonces voy a colocar el nombre, vamos a poner así nombre, puedo poner Anahí Salgado, recuerda que solamente estos son textos de muestra. Teléfono podemos colocar 555777.

Vamos a colocar el email, el email, y por ejemplo puedo colocar Anahí@gmail.com.

Esto simplemente son datos de muestra que los vamos a utilizar en Activity detalleContacto.

Vamos por aquí a colocarle un texto android:text, y vamos a usar el recurso @string, el recurso nombre, el string nombre.

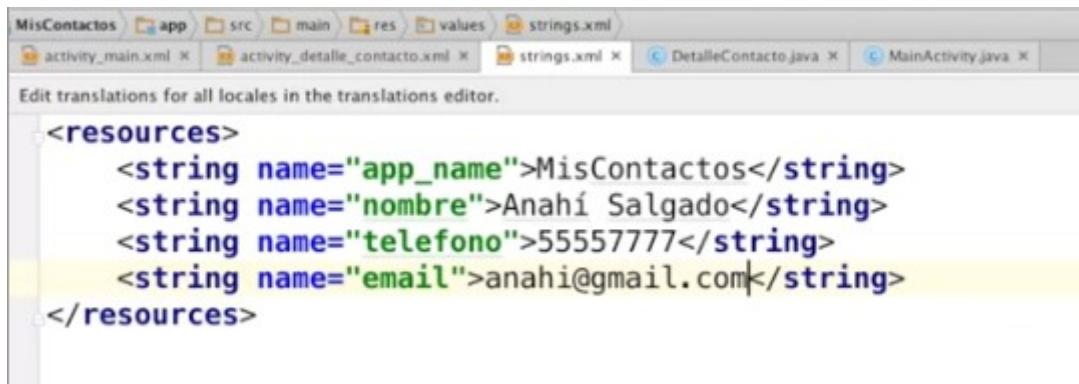
Aquí ya se alcanza a ver el nombre, más abajo colocaré un TextView y voy a colocar wrap_content, wrap_content, su Id, recuerda que este será el teléfono.

Vamos a poner aquí TextView, teléfono, muy bien.

Colocamos nuestro android:text@string.

Escribir en, en esta vista de código está bien porque puedo tener mayor control del código, pero si en, en algún momento se te facilita más utilizar la vista de diseño con toda libertad puedes hacerlo.

Básicamente es lo que más, lo que más se te adapte, lo que más te adaptes, y lo que más puedas sobre todo desarrollar más rápido.



```
<resources>
    <string name="app_name">MisContactos</string>
    <string name="nombre">Anahí Salgado</string>
    <string name="telefono">55557777</string>
    <string name="email">anahi@gmail.com</string>
</resources>
```

Y en activity_detalle_contacto:

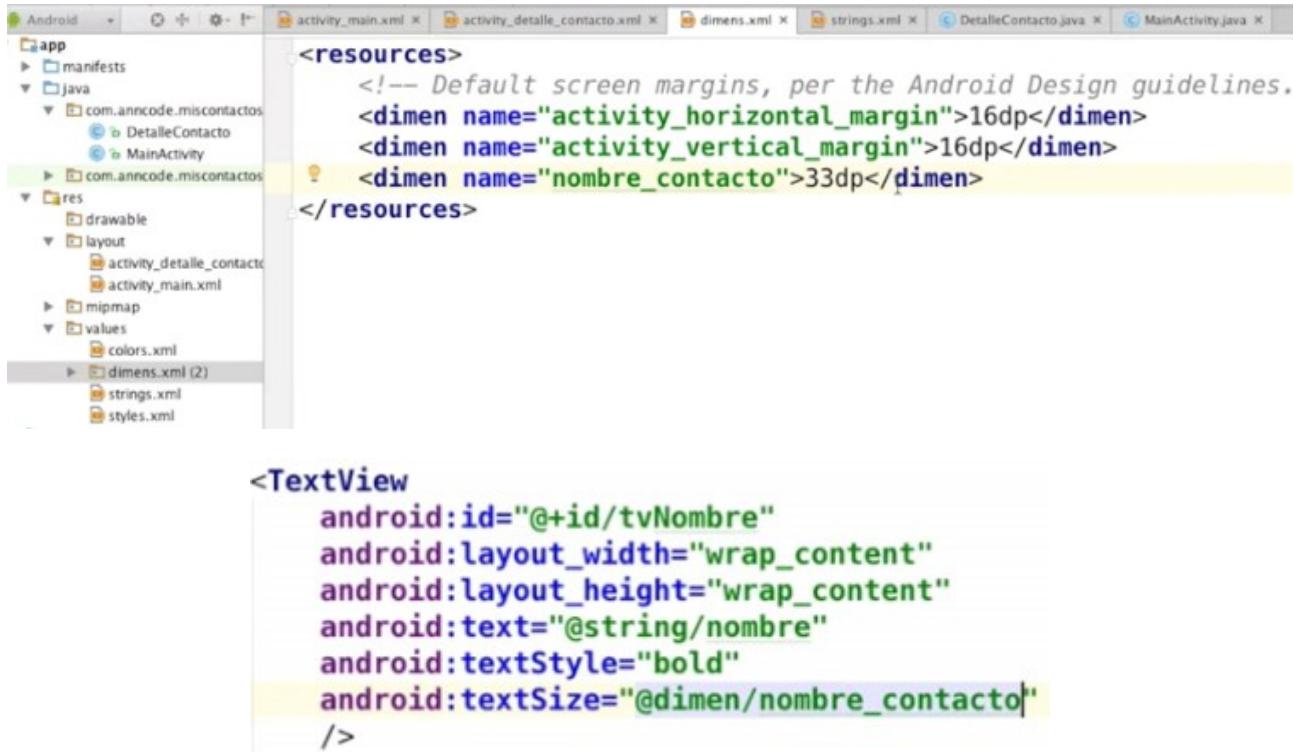
```
<TextView
    android:id="@+id/tvTelefono"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/telefono"

/>
<TextView
    android:id="@+id/tvEmail"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/email"/>
```

Bien, el email vamos a ponerlo el texto @string email, aquí está.

Perfecto, ahora esto pues no tiene un formato pues muy agradable, además dijimos que ibamos a colocar aquí una imagen con que tendrá nuestro icono, y a un lado también el icono para el correo electrónico y para el teléfono, entonces primeramente voy a concentrarme en el nombre.

El nombre lo voy a hacer un poco más grande y además voy a colocar en negritas, es decir, vamos a darle más formato android, vamos a colocarlo primero en negritas con la propiedad TextStyle, vamos a colocar bold. Vamos a poner una, un texto más grande con la propiedad TextSize, a lo mejor si colocamos unos 20 dp, a los mejor un poco más o un poco menos, yo creo que algo así está bien, es más legible. Bien, esta medida la voy a quitar de aquí y voy a declararla en mi archivo dimens.



The screenshot shows the Android Studio interface. On the left is the Project Navigational Drawer, which lists the project structure: app (manifests, java, com.anncode.miscontactos (DetalleContacto, MainActivity), res (drawable, layout (activity_detalle_contacto, activity_main.xml), mipmap, values (colors.xml, dimens.xml (2), strings.xml, styles.xml). The right side shows the code editor with the dimens.xml file open. The code defines margins and a dimension for the contact name:

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="nombre_contacto">33dp</dimen>
</resources>
```

Below this, the activity_main.xml layout file is shown, containing a TextView element with the following attributes:

```
<TextView
    android:id="@+id/tvNombre"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/nombre"
    android:textStyle="bold"
    android:textSize="@dimen/nombre_contacto" />
```

Ahora vamos a tratar la información que está aquí, el teléfono y el correo electrónico.

Primeramente pues para que mi imagen pueda salir a la derecha, y a un lado el teléfono, por lo que necesitaré colocar un layout anidado en este layout, o sea layout dentro, un layout como hijo de este LinearLayout.

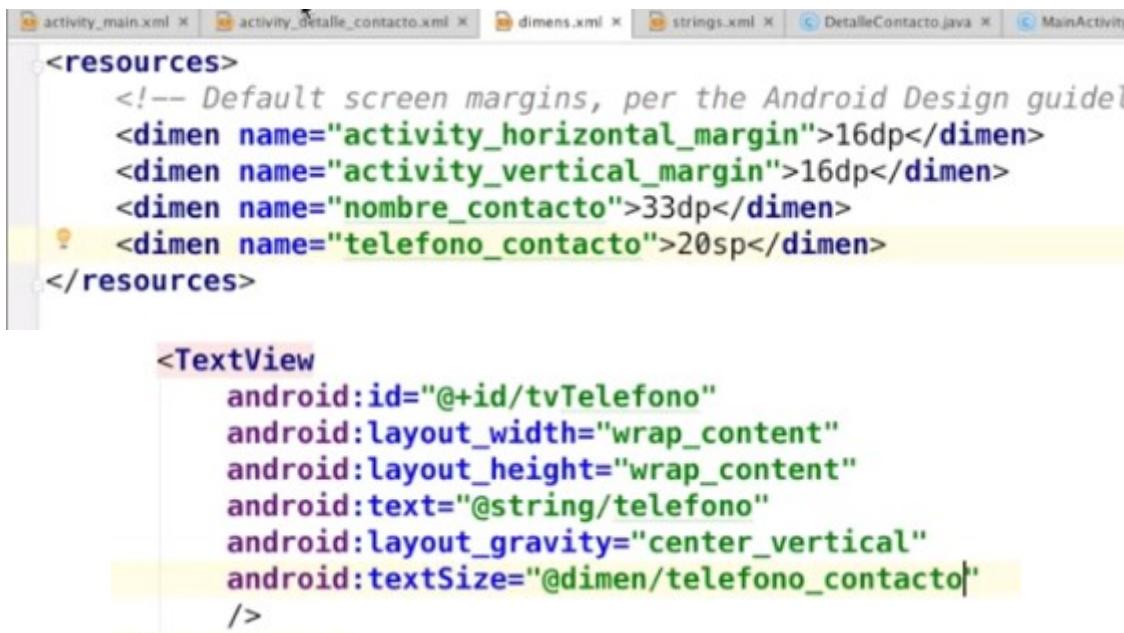
Un layout que me va a ayudar pues a manejar mejor, el layout que me ayudará mejor será otro LinearLayout que voy a colocar la posición horizontal, para que la imagen quede de un lado y el texto quede de otro.

Voy a colocar aquí el LinearLayout que en su width esté como match_parent y en su height esté como wrap_content. Bien, aquí y vamos a meter este TextView que pertenece al teléfono, lo vamos a meter acá, listo, ahí está, no olvidemos la orientación del LinearLayout es una propiedad muy importante y vamos a decir que sea horizontal. Ahora vamos a colocar aquí un elemento ImageView.

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal"  
>  
  
<ImageView  
    android:id="@+id/imgvTelefono"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/phone_48"  
>  
  
<TextView  
    android:id="@+id/tvTelefono"  
    android:layout_width="wrap_content"
```

Vamos a colocar en nuestro ImageView, nuestro ImageView vamos a poner nuestro ícono que dejamos por aquí, vamos a poner android:src y con @drawable coloco el ícono del teléfono.

Bien, ya se ve aquí, se alcanza a ver mi teléfono. Vamos a ver si por aquí tenemos una propiedad disponible para poder centrar este elemento verticalmente y que quede, pues que quede alineado con el ícono.



```
<resources>  
    <!-- Default screen margins, per the Android Design guidelines -->  
    <dimen name="activity_horizontal_margin">16dp</dimen>  
    <dimen name="activity_vertical_margin">16dp</dimen>  
    <dimen name="nombre_contacto">33dp</dimen>  
    <dimen name="telefono_contacto">20sp</dimen>  
</resources>  
  
<TextView  
    android:id="@+id/tvTelefono"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/telefono"  
    android:layout_gravity="center_vertical"  
    android:textSize="@dimen/telefono_contacto"  
>
```

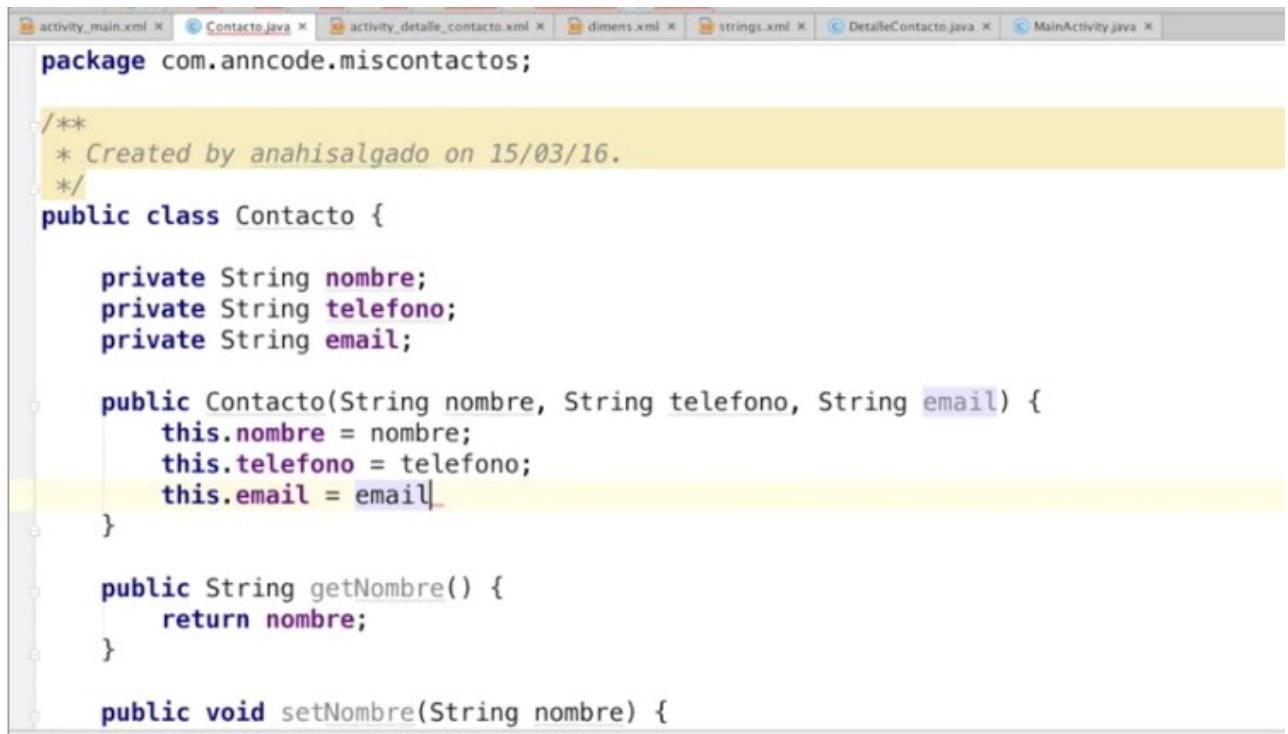
Por otra parte se define el tamaño del texto en sp para que no dependa de la resolución y se eligió ese tamaño para que quepa el dedo al hacer click ahí.

Lo mismo se hace para el email.

Ahora vamos a la pantalla de inicio (main activity.java)

Aquí lo que se quiere mostrar es un ArrayList de contactos, por lo que se crea una nueva clase que define a los contactos como objetos.

Ésto lo que genera es, por un lado el constructor, dado que un contacto debe tener sí o sí un nombre, un número de teléfono y un email; y por otro lado los métodos setters y getters.



The screenshot shows the Android Studio interface with the 'Contacto.java' file open in the center. The file contains Java code for a 'Contacto' class. It includes private fields for name, phone, and email, a constructor that initializes these fields, and methods to get and set the name. The code is color-coded for readability, with comments in green and class names in blue.

```
package com.anncode.miscontactos;

/**
 * Created by anahisalgado on 15/03/16.
 */
public class Contacto {

    private String nombre;
    private String telefono;
    private String email;

    public Contacto(String nombre, String telefono, String email) {
        this.nombre = nombre;
        this.telefono = telefono;
        this.email = email;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
```

Y en el mainactivity.java:

```

activity_main.xml X Contacto.java X activity_detalle_contacto.xml X dimens.xml X strings.xml X DetalleContacto.java X MainActivity.java X

package com.anncode.miscontactos;

import ...

public class MainActivity extends AppCompatActivity {

    ArrayList<Contacto> contactos;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        contactos = new ArrayList<Contacto>();

        contactos.add(new Contacto("Anahi Salgado", "77779999", "anahi@gmail.com"));
        contactos.add(new Contacto("Pedro Sanchez", "88882222", "pedro@gmail.com"));
        contactos.add(new Contacto("Mireya Martinez", "33331111", "mireya@gmail.com"));
        contactos.add(new Contacto("Juan Lopez", "44442222", "juan@gmail.com"));

    }

    ListView lstContactos = (ListView) findViewById(R.id.lstContactos);
    lstContactos.setAdapter(new ArrayAdapter<>());
}

lstContactos = (ListView) findViewById(R.id.lstContactos);
tos.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, contactos));

```

Ahora, Si quiero solo los nombres:

```

ArrayList<String> nombresContacto = new ArrayList<>();
for (Contacto contacto : contactos) {
    nombresContacto.add(contacto.getNombre());
}

lstContactos = (ListView) findViewById(R.id.lstContactos);
lstContactos.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, nombresContacto))

```

Ahora, ésto imprime en pantalla la lista de nombres, pero para que vaya a la pantalla de Detalle de contacto, hay que hacer un intent explícito.

Pero antes, hay que colocar un listener que capte donde se hace click.

SetOnItemClickListener, le voy a colocar aquí new y me pide la interfaz onItemClickListener, se la voy a dar, y automáticamente todo el código que necesito me lo sobreescribe.

```

lstContactos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Intent intent = new Intent(MainActivity.this, DetalleContacto.class);
        startActivityForResult(intent);
    }
});

```

Aquí está mi método onItemClick, el método onItemClick va a estar capturando algunos parámetros interesantes que en primer lugar estaría cachando la posición, la posición del elemento en el cual estoy obteniendo. Por ejemplo en este caso Anahí Salgado me cachará la posición 0.

Ahora, dentro de onItemClick es donde se coloca el intent, con 2 parámetros, el primero es dónde estoy y el segundo es hacia donde voy. Y en el siguiente renglón se inicia el intent.

Pero ésto no hace que cambie para todos los contactos, solo refleja en la pantalla para uno que es Anahí Salgado.

Para modificar éste comportamiento, se usa el método putExtra:

```
lstContactos.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
        Intent intent = new Intent(MainActivity.this, DetalleContacto.class);  
        intent.putExtra("Nombre", contactos.get(position).getNombre());  
        intent.putExtra("Telefono", contactos.get(position).getTelefono());  
        intent.putExtra("Email", contactos.get(position).getEmail());  
        startActivity(intent);  
    }  
});
```

Bien, y aquí también una opción que me da, intent.putExtra es que puedo enviar un arreglo, puedo enviar un arreglo de cualquier elemento primitivo un arreglo e incluso de un string que no es un tipo primitivo pero puedo enviar arreglos de string, entonces otra forma también más óptima es colocar a lo mejor estos elementos en un string y enviar todo el string de parámetros, como un solo parámetro.

Lo vamos a coloca así para que puedas ver cómo podemos mandar varios parámetros en un cambio de pantalla.

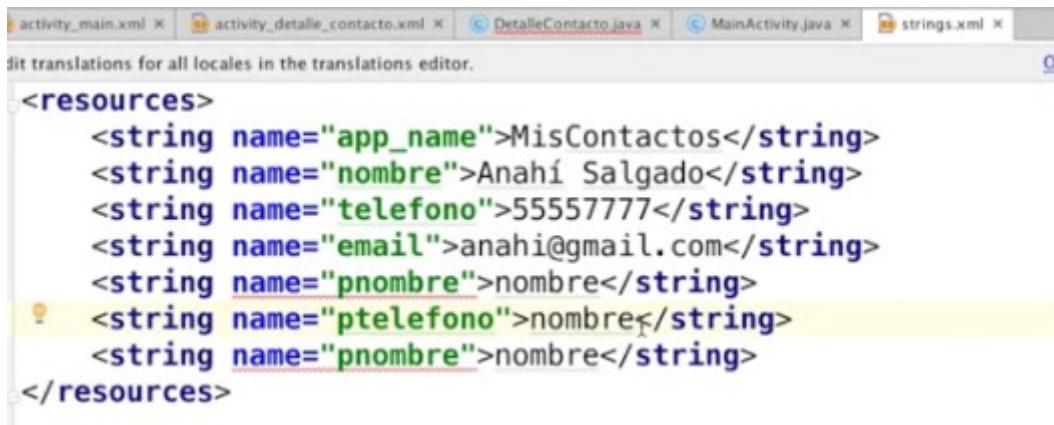
Bien, entonces ya tenemos aquí, esto es un intent explícito, un intent explícito que nos permite unir componentes una pantalla a otra.

Si se quiere hacer más general:

En el mainactivity.java:

```
lstContactos.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
        Intent intent = new Intent(MainActivity.this, DetalleContacto.class);  
        intent.putExtra(getResources().getString(R.string.pnombre), contactos.get(position).getNombre());  
        intent.putExtra(getResources().getString(R.string.ptelefono), contactos.get(position).getTelefono());  
        intent.putExtra(getResources().getString(R.string.pemail), contactos.get(position).getEmail());  
        startActivity(intent);  
    }  
});
```

Y en el archivo strings.xml:



```
<resources>
    <string name="app_name">MisContactos</string>
    <string name="nombre">Anahí Salgado</string>
    <string name="telefono">55557777</string>
    <string name="email">anahi@gmail.com</string>
    <string name="pnombre">nombre</string>
    <string name="ptelefono">nombre</string>
    <string name="pnombre">nombre</string>
</resources>
```

Ahora del otro lado, del detalleContacto, lo que yo tengo que hacer es que una vez que estoy mandando los parámetros, tengo que recibir los parámetros.

Y si recuerdas cómo recibir los parámetros, lo hago un objeto bundle.

Un objeto bundle que se puede llamar parámetros, y que a este objeto le solicitaré el intent, si es que viene un intent, getExtras y recuerda que en Android los parámetros se llaman extras.



```
package com.anncode.miscontactos;

import ...

public class DetalleContacto extends AppCompatActivity {

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

        Bundle parametros = getIntent().getExtras();
        String nombre = parametros.getString(getResources().getString(R.string.pnombre));//non
        String ngmbre = parametros.getString(getResources().getString(R.string.pnombre));//non
        String nombre = parametros.getString(getResources().getString(R.string.pnombre));//non
    }
}
```

Ya tengo los datos, simplemente ahora vamos a mostrarlos respectivamente. Vamos a mostrarlos en nuestros views que tenemos por acá.

Tenemos nuestro TextView, tenemos nuestro TextView teléfono, nuestro TextView nombre, todos, todos y cada uno tienen un identificador que me va a ayudar a poder trabajarlos con código Java.

Entonces voy a declarar mis TextViews. Vamos a poner tvNombre, TextView findViewById R.Id.TextViewNombre.

```
public class DetalleContacto extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_detalle_contacto);  
  
        Bundle parametros = getIntent().getExtras();  
        String nombre = parametros.getString(getResources().getString(R.string.pnombre));//n  
        String telefono = parametros.getString(getResources().getString(R.string.ptelefono))  
        String email = parametros.getString(getResources().getString(R.string.pemail));//ema  
  
        TextView tvNombre = (TextView) findViewById(R.id.tvNombre);  
        TextView tvTelefono = (TextView) findViewById(R.id.tvTelefono);  
        TextView tvEmail = (TextView) findViewById(R.id.tvEmail);  
  
        tvNombre.setText(nombre);  
        tvTelefono.setText(telefono);  
        tvEmail.setText(email);  
    }  
}
```

Silenciar

Lo mismo para el teléfono y para el email. Podemos correrlo y ahora que yo you esté dando clic en algún elemento de la lista, la idea es que pues ya se estén mostrando esos elementos también.

Únicamente nos falta hacer que cuando yo presione el teléfono o presione el correo electrónico pues, estos tengan, ejecute una llamada y levante una aplicación de correo electrónico.

Ahora lo que haremos será precisamente pues darle vida al teléfono y al email, y para ello crearé, crearé dos métodos.

Uno para ejecutar una llamada public_void, llamar, aquí recibiremos un objeto view, sí, y más abajo colocaremos public_void email, enviarMail.

Vamos a colocar nuestro view, así.

Bien, entonces antes de hacer estos métodos, solamente los tengo aquí declarados, estos TextView los voy a declarar globales para poder manipularlos dentro de cada método.

```
public class DetalleContacto extends AppCompatActivity {

    private TextView tvNombre;
    private TextView tvTelefono;
    private TextView tvEmail;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detalle_contacto);

        Bundle parametros = getIntent().getExtras();

        String nombre      = parametros.getString(getResources().getString(R.string.pnombr));
        String telefono   = parametros.getString(getResources().getString(R.string.ptele));
        String email       = parametros.getString(getResources().getString(R.string.pemai));

        tvNombre     = (TextView) findViewById(R.id.tvNombre);
        tvTelefono   = (TextView) findViewById(R.id.tvTelefono);
        tvEmail      = (TextView) findViewById(R.id.tvEmail);

        tvNombre.setText(nombre);
        tvTelefono.setText(telefono);
        tvEmail.setText(email);
    }
}
```

En nuestro layout tenemos esto, tenemos el teléfono y tenemos el correo.

La idea es que cuando yo le de clic aquí, ejecute la llamada, cuando le de clic acá se ejecute el correo.

Podemos cachar aquí en nuestro layout desde aquí, desde el código xml, una propiedad onClick.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginTop="@dimen/top"
    android:onClick="llamar"
>
```

Ésto hace que lo que esté dentro de ese linear layout, al presionarlo, le de llamar. Y lo mismo se hace para email.

Vamos a hacer que al llamar, lo primero que yo necesito obtener para llamar para ejecutar la llamada es el número telefónico,

eso es lo primero que necesito obtener.

Entonces utilizaré mi tvTelefono y vamos a ponerle .getText, .getText.toString.

Con el .getText no es suficiente porque el .getText nos trae un objeto tipo char sequence, al hacer yo un toString en ese momento, ya obtengo algo útil, teléfono, y entonces lo puedo traer aquí.

Bien, una vez obtenido el teléfono en esta variable local, es hora de ejecutar mi intent.

Para ejecutar mi intent, lo que estaré ejecutando, el tipo de intent que estaré ejecutando será un intent implícito, porque será un intent que tome un recurso externo de mi aplicación, es decir para que se pueda ejecutar la llamada yo tengo que abrir la aplicación de llamadas y enlazarla con mi aplicación, entonces este es un intent implícito.

Para hacer un intent implícito, haremos una instrucción muy sencilla, startActivity, y aquí adentro colocaremos nuestro intent implícito.

Podría declararlo como lo estuvimos haciendo anteriormente, pero esta es una forma mucho más sencilla.

Recuerda que el intent implícito siempre va a ejecutar una acción, una acción del sistema.

Entonces el primer parámetro será intent.action y aquí lo que voy a buscar, la acción que necesito es la acción call, la acción de llamada. Posteriormente el segundo parámetro es el número telefónico.

No puedo pasar el número telefónico tal cual lo estoy obteniendo aquí, sino que el número telefónico lo tengo que hacer, lo tengo que manejar como un recurso accesible, un recurso accesible, un recurso de tipo Uri.

Un Uri, ejecutaremos el método .parse, colocaremos nuestro teléfono y el Uri.parse deberá tener por acá consecutivo la palabra tel.

```
public void llamar(View v){  
    String telefono = tvTelefono.getText().toString();  
    startActivityForResult(new Intent(Intent.ACTION_CALL, Uri.parse("tel:" + telefono)));  
}
```

En este momento nos está alertando el IDE que necesito tener permisos, tener declarado el permiso de call_phone, esto en el AndroidManifest.

Esto lo vamos a ver en secciones más adelante, sobre cómo manejar la seguridad del Android, pero mientras tanto declaremos los permisos que nos está solicitando.

Vamos a abrir nuestro AndroidManifest, lo tenemos por aquí y vamos a colocar uses-permission y vamos a colocar el permiso de call_phone.

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.anncode.miscontactos">  
  
    <uses-permission android:name="android.permission.CALL_PHONE">
```

Luego simplemente voy a dar alt enter, y vamos a hacer que nos genere un código donde estará chequeando si el permiso está dado de alta o si el permiso está asignado.

Esto será en modo de run time, en modo mientras la aplicación está corriendo, se estará verificando que realmente se tenga, el usuario haya dado acceso a este permiso para nuestra aplicación.

Entonces, con esto, ya no debería marcarnos ningún error y ya tendríamos nuestro intent implícito para ejecutar una llamada.

```
public void llamar(View v) {
    String telefono = tvTelefono.getText().toString();
    if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
        // TODO: Consider calling
        // ActivityCompat#requestPermissions
        // here to request the missing permissions, and then overriding
        // public void onRequestPermissionsResult(int requestCode, String[] permissions,
        //                                         int[] grantResults)
        // to handle the case where the user grants the permission. See the documentation
        // for ActivityCompat#requestPermissions for more details.
        return;
    }
    startActivity(new Intent(Intent.ACTION_CALL, Uri.parse("tel:" + telefono)));
}
```

Para enviar un email también ejecutaremos un intent implícito.

Entonces, de la misma forma que con el teléfono necesitaré la dirección de correo electrónico y la obtendré a partir del TextView, getText.toString.

Y ahora ya tengo el email, ya tengo el email de la persona y entonces puedo comenzar a ejecutar mi intent. Vamos a poner start, aquí sí necesitaré declarar un objeto de tipo intent, emailIntent.

Le vamos a poner new igual a intent, y la acción que necesitaremos aquí será Intent.Action_Send, Lo siguiente que debo colocar en la configuración de mi intent será setData, setData, que el setData lo que tendrá simplemente es la dirección Uri de lo que va a suceder, Uri.parse, y vamos a decir que sera un mailTo.

Será esta la acción de lo que va a suceder, o sea qué tipo de envío será, en este caso mailto nos indica que será un envío de email.

Y entonces ahora procederemos a definir parámetros, parámetros que serán unos parámetros que ya están previamente definidos en esta estructura de email, y entonces voy a colocar .putExtra, puedo colocar mi primer parámetro Intent.Extra_email, Extra_email.

Con esto, decidí a quién será enviado el correo, ¿a quién será enviado el correo? Pues básicamente es a la variable, a lo que estamos tomando, el dato que tenemos aquí, email.

```
public void enviarMail(View v){
    String email = tvEmail.getText().toString();
    Intent emailIntent = new Intent(Intent.ACTION_SEND);
    emailIntent.setData(Uri.parse("mailto:" + email));
    emailIntent.putExtra(Intent.EXTRA_EMAIL, email);
}
```

Yo puedo tener aquí más opciones, puedo tener en este caso Extra_email, puedo tener Extra_CC si quiero posteriormente también enviar una copia, puedo colocar también el subject, si quiero asignar un asunto al correo electrónico y además con la propiedad Text puedo definir un cuerpo previo para el correo electrónico.

Para el caso de mi aplicación no es necesario ninguno de estas variables que tengo disponibles, sino la única que necesito es el del Email porque simplemente va a abrir una aplicación de correo electrónico y me va a precargar el email dejando todos los demás parámetros libres para el usuario, para que él coloque lo que quiera.

Entonces, ahora sí, startActivity emailIntent, y con esto coloco startActivity y dentro deberé colocar un chooser o una aplicación que me va a permitir, se va a levantar, que me va a permitir elegir entre todas mis aplicaciones de correo que tengo disponibles para abrir, entonces voy a colocar createChooser y ahí dentro voy a poner mi emailIntent, y a ese chooser simplemente vamos a colocarle un título Email.

```
public void enviarMail(View v){  
    String email = tvEmail.getText().toString();  
    Intent emailIntent = new Intent(Intent.ACTION_SEND);  
    emailIntent.setData(Uri.parse("mailto:"));  
    emailIntent.putExtra(Intent.EXTRA_EMAIL, email);  
    startActivityForResult(Intent.createChooser(emailIntent, "Email"));  
}  
  
Intent target, CharSequence title  
Intent target, CharSequence title, IntentS
```

Con setType voy a indicar qué tipo de aplicación o qué tipo de identificador de aplicaciones o en qué categorías se encuentran las aplicaciones de email.

Con esto estoy definiendo que lo que quiero que me coloque como un chooser, sea todas las aplicaciones de email.

Volvemos a correr nuestro proyecto, y entonces cuando yo abro, aquí están todas las aplicaciones que tengo disponibles para enviar un mail.

```
public void enviarMail(View v){  
    String email = tvEmail.getText().toString();  
    Intent emailIntent = new Intent(Intent.ACTION_SEND);  
    emailIntent.setData(Uri.parse("mailto:"));  
    emailIntent.putExtra(Intent.EXTRA_EMAIL, email);  
    emailIntent.setType("message/rfc822");  
    startActivityForResult(Intent.createChooser(emailIntent, "Email "));  
}
```

Terminando Activities

El problema de todo lo que vimos anteriormente, es que si se tienen varias actividades, quedan todas apiladas y, si por ejemplo hay 20 actividades y se presiona el botón de home, todas ellas van al background, lo cual puede ocupar mucho espacio en memoria RAM.

Para eso, se puede hacer que, al comenzar una actividad nueva, se cierre la anterior para que no se quede acumulada en la pila, lo cual se logra en la actividad, luego de cargar el intent, dar la instrucción finish(), en éste caso, en el activitymain.java

```
lstContactos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Intent intent = new Intent(MainActivity.this, DetalleContacto.class);
        intent.putExtra(getResources().getString(R.string.pnombre), contactos.get(position));
        intent.putExtra(getResources().getString(R.string.ptelefono), contactos.get(position));
        intent.putExtra(getResources().getString(R.string.pemail), contactos.get(position));
        startActivity(intent);
        finish();
    }
});
```

Ésto hace que, si estoy en la segunda actividad y aprieto el botón de back, la aplicación va a terminar.

Yo puedo estar tomando también el evento de regresar, es decir, cuando alguien presione una tecla y puedo hacerlo con el método on key, on key, on key down, press. On key down. Y lo que voy a estar recibiendo ahí es un código del código, un código de la tecla presionada,

Y además, un evento. Entonces yo aquí podría comparar. Y entonces lo que estaré comparando será el key code, el key code que estoy tomando, y lo estaré comparando con el objeto con key event y entonces yo podré decir que si el key code que estoy tomando es el key code de back, por ejemplo, corresponde a la tecla de back.

Entonces, lo que te indica hacer aquí es levantar un nuevo intent. Levantar un nuevo intent que ahora vaya en este caso a la lista, porque estoy en el detalle, ¿no?

Entonces, debo colocar un intent y le pondré intent igual new intent, colocaré this y en dónde estoy ubicado y a dónde quiero ir. Quiero ir a MainActivity.class. MainActivity.class y simplemente lanzo la actividad.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK){
        Intent intent = new Intent(DetalleContacto.this, MainActivity.class);
        startActivity(intent);
    }
    return super.onKeyDown(keyCode, event);
}
```

Ahora sí, entonces yo puedo tener control de si una actividad es eliminada y ahora puedo tener control para poder regresar a la actividad anterior, para que no se quede vacío el start.

Si yo entro en este momento, la actividad anterior se ha eliminado y si regreso, me regresa a la actividad donde estaba.

Entonces, esta es una buena práctica, es una buena forma de poder estar optimizando nuestras aplicaciones cuando de pronto se vuelven muy pesadas en nuestro teléfono

Muchos usuarios, a mucha gente, no nos gusta que de pronto una aplicación haga lento o ralentice nuestro teléfono. Entonces esto es una forma de poder optimizar nuestras aplicaciones para que consuman menos recursos.

Semana 3

RecyclerView

Acabamos de crear una aplicación de contactos donde tenemos una lista con todos nuestros contactos, pero, ¿cómo podemos hacer más eficiente y sobre todo también más legible con los nuevos elementos de material design nuestra aplicación?

Bien. Existe un nuevo elemento que se llama RecyclerView, RecyclerView es un nuevo elemento que se ha introducido en material design, es un nuevo widget, y es utilizado principalmente para manejar listas de datos.

RecyclerView pretende sustituir por completo al famoso ListView. ¿Esto por qué? Porque RecyclerView como su nombre lo dice, Recycler de reciclaje, hará un reciclaje con todos tus items, es decir será mucho más eficiente.

¿Cómo ejecuta este reciclaje? Supongamos que tenemos una lista de unos 100 elementos, si estos 100 elementos los estamos mostrando en un ListView, bueno, todos los 100 elementos automáticamente se están cargando en la vista haciendo que de pronto esa pantalla de interfaz de usuario sea bastante pesada.

Con RecyclerView, solamente como estamos viendo en esta pantalla, solamente estaremos cargando los datos que se ven en pantalla. Y para poder seguir viendo los datos anteriores, estaremos reciclando un layout, estaremos reciclando un item en particular.

Imagina esta lista como un conjunto de renglones, cada renglón va a tener, si fuese una lista de contactos podría tener una foto de nuestro contacto, podría tener su nombre y a lo mejor un teléfono que sea directo para que nos de un preview de ese contacto.

Si nosotros manejamos todo esto es un ListView, estaremos manejándolo sí en un layout pero en un layout que se estará repitiendo 100 veces, o dependiendo de la cantidad de elementos que tengas.

Para un RecyclerView no tendremos los 100 elementos acumulados, sino que solamente tendremos en pantalla los 5 elementos que quepan, o los 10, o los 2 elementos que quepan y cuando el usuario haga scroll sobre la pantalla, en ese momento los elementos serán mostrados y además estará habiendo un reciclaje de elementos.

De esta forma las interfaces no son tan pesadas en comparación de utilizar un ListView.

Si utilizamos un RecyclerView estará reciclando nuestros layouts, estará reciclando nuestras vistas para hacer esto mucho más eficiente.

Una de las ventajas también de utilizar RecyclerView, es que los elementos no solamente pueden ser mostrados en una lista, también pueden ser mostrados en forma de grid o de una forma escalonada utilizando un LayoutManager para cada uno de estos.

Vamos a ver más adelante cómo podemos estar manejando un RecyclerView.

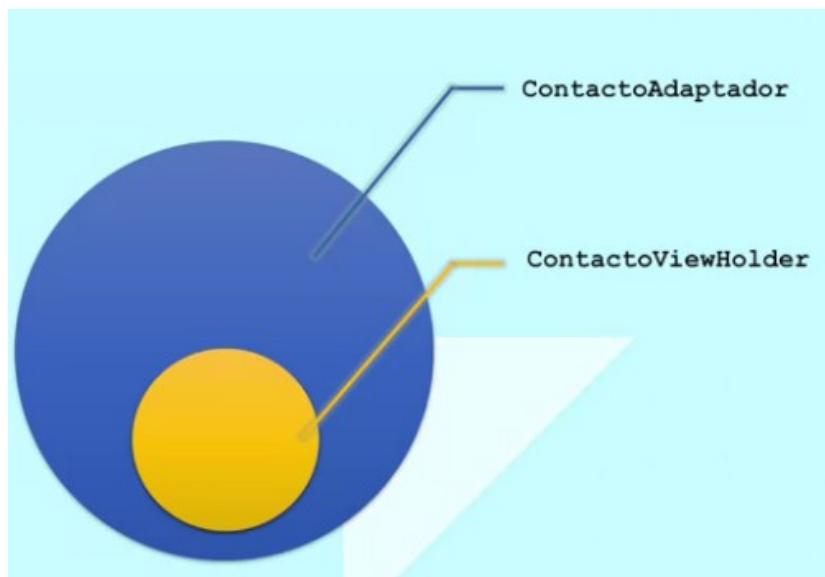
La composición de un RecyclerView se puede ejemplificar con la siguiente figura.



Primeramente tenemos un RecyclerView y un LayoutManager, este me va a ayudar precisamente a darle forma a mi lista, cómo es que quiero mostrar mi lista, si quiero mostrar los elementos en forma de grid, en forma escalonada o apilada o simplemente en una lista. Para eso utilizaré la clase auxiliar LayoutManager.

Ahora, si hablamos sobre el adaptador, el adaptador podría contener una clase anidada, es decir tendremos nuestra clase normal, nuestra clase adaptador para el caso de nuestra lista de contactos, tendríamos nuestra clase ContactoAdaptador.

Esta clase ContactoAdaptador podría tener una clase anidada que comúnmente es el nombre de la clase extensión ViewHolder.



ViewHolder nos va a ayudar a manejar toda la conexión entre el adaptador y todos nuestros elementos views que componen el renglón de tu RecyclerView o el elemento que quieras mostrar en el RecyclerView, ViewHolder.

Posteriormente tenemos nuestra clase adaptador. La clase adaptador que es una clase que está fuera de la clase anidada ViewHolder y esta clase adaptador va a ayudar a que podamos sincronizar todos los views que tenemos por un lado, los views de nuestro renglón con los datos obtenidos de cualquier lugar.

Los datos puedes tenerlos en un Web Service o en una base de datos como ya dijimos, pero básicamente aquí es donde estaremos manejando la lista de datos.

Esta clase de adaptador contendrá tres métodos importantes:

- **onCreateViewHolder**, onCreateViewHolder como estamos observando aquí, se va a encargar precisamente de tomar ese layout, ese layout que define cada elemento de toda tu lista de RecyclerView, va a tomar el layout y lo va a sincronizar con nuestro ViewHolder, nuestra clase anidada que tenemos ahí mismo.
- **onBindViewHolder**, este método onBindViewHolder se va a encargar de setear o de colocar todos los datos de una lista o de una colección de datos que puedes traer en un elemento de tipo list o en un ArrayList o en un vector o lo que tú quieras, se va a encargar precisamente de tomar esa lista, setearla y además irla asociando a cada view de nuestro ViewHolder.
- **getItemCount**, el método getItemCount lo único que nos va a devolver es el resultado de toda la cantidad de elementos que tenemos, es decir cuántos elementos estoy manejando en la lista. Para nuestro ejemplo nos devolverá la cantidad de contactos que estoy teniendo, esto para tener un mejor control sobre la lista y poder hacer su trabajo de reciclaje.

Dataset, Dataset es precisamente el POJO que acabamos de crear. El POJO nos va a permitir manejar toda la interacción, si vas a traer los datos de un Web Service o si vas a traer los datos de una base de datos, etcétera, el POJO nos va a permitir manejar toda esta entidad de datos, para posteriormente comunicársela al adaptador y una vez que se tenga esa comunicación de adaptador, mostrarlo finalmente en un layout.



Este es el gráfico, esta es la composición para lograr un RecyclerView. Si queremos ver esto a nivel de código, si queremos mostrar nuestros elementos en una lista, utilizaremos la clase LinearLayoutManager.

Si queremos ver nuestros elementos en forma de grid, utilizaremos la clase GridLayoutManager. Y si queremos ver nuestros elementos de forma apilada, utilizaremos la clase StaggeredGridLayoutManager.

Todo esto debe estar concentrando en el Activity donde vas a mostrar tu RecyclerView. Podría ser en tu MainActivity y recuerda, todo va dentro de tu método onCreate.

Ejemplo de RecyclerView

Ok, en este momento me encuentro en el ejercicio que acabamos de hacer sobre nuestra lista de contactos. A continuación vamos a hacer que esta lista se parezca más a un recycler view tal cual como acabamos de ver en la sección anterior cuando acabamos de ver toda la explicación.

Vamos a pasar este List view, estos elementos a un recycler view.

En secciones anteriores cuando estuvimos hablando sobre material design, vimos cómo realizar uno de los nuevos elementos, que son los llamados card views.

Un recycler view, precisamente debía ser que un card view pueda estarse reciclando repetidas veces.

Y que de alguna forma podamos tener una vista en forma de una lista o en forma de un grid o en forma escalonada. Es decir, como si estuvieran revueltos los elementos, pero bien acomodados.

Entonces vamos nosotros a hacer un card view, vamos a hacer un en una tarjeta.

Esta tarjeta será nuestro layout base, el cual estará reciclando gracias a nuestro recycler view. Y entonces de esta forma es como cada contacto que tenemos en nuestra lista, lo estaremos plasmando en un layout que será nuestro card view.

Y así mismo pues se estará reciclando para generar finalmente nuestra lista de contactos.

Entonces tengo esta nota por aquí, donde lo que voy a hacer es simplemente empezar a modelar un poco cómo es que queremos que se vea nuestra tarjeta.



Esto a veces suele ser una buena práctica y suele ser algo útil cuando estamos desarrollando interfaces que a lo mejor se vuelven un poco complicadas, en cuanto a términos de xml y de layouts.

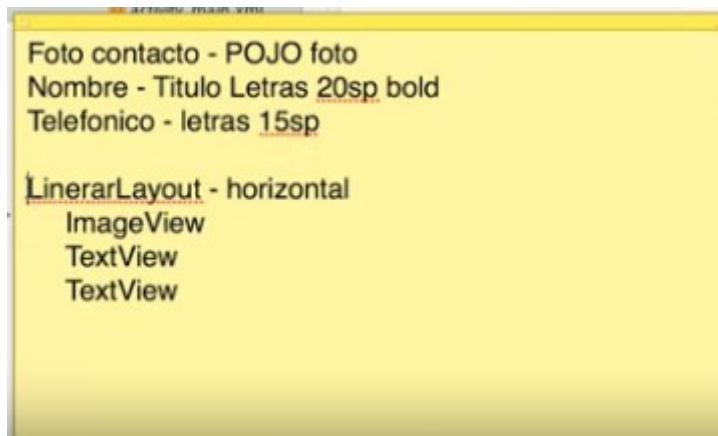
Entonces pues vamos a diseñar un poquito, cómo es que vamos a va quedar nuestra tarjeta.

Primeramente lo que quiero que se vea en mi tarjeta es la foto de un contacto.

Ésto va involucrar que yo tenga que agregar en mi POJO, que hicimos nuestro POJO de contactos, un nuevo atributo. Que será precisamente realmente pues la foto.

Posteriormente eso es lo que yo quisiera mostrar, mi foto y después quisiera ver el nombre del contacto. A lo mejor el nombre del contacto que sea en forma como de título, es decir, la letras que estén como en unos 20sp, y esté además en negritas o en bold.

Y bueno, consecutivamente podría yo colocar a lo mejor el número telefónico. El número telefónico que sea un texto pequeño que a lo mejor las letras sean como de unos 15sp. Esta no tendría un bold porque me interesa únicamente resaltar el nombre.



Entonces de acuerdo a esta estructura que quiero que tenga mi tarjeta, puedo ya comenzar a definir el layout que quisiera que manejara estos elementos. Que agrupara y ordenara estos elementos.

Entonces podría yo comenzar por un LinearLayout. Un LinearLayout que sea. Yo quiero que estos elementos estén uno debajo de otro, consecutivamente. Entonces podría colocar un LinearLayout que sea horizontal y bueno todos los elementos.

Aquí tendrá el elemento final, la foto de contacto será un ImageView. Consecutivamente el elemento que defina el nombre será un TextView y el nombre que defina el teléfono también será un TextView.

Entonces este layout, esta forma será lo que defina mi card view, mi elemento de tarjeta que va estar duplicándose y que va estar reciclando para generar los demás elementos.

Entonces, como dijimos pues voy a añadir un nuevo elemento a mi foto, a mi POJO, el nuevo elemento de foto.

Estoy aquí en el POJO de contacto. Y vamos a manejar la foto como un número entero. Puesto que por el momento nuestras fotos y nuestras imágenes también van a estar viviendo en la carpeta de drawable.

Entonces voy a colocar private int foto.

Y entonces también quiero por aquí, que el constructor tenga la foto. Aquí le voy a pasar, voy a pasar por aquí la foto. This.foto = foto.

```

/*
 * Created by anahisalgado on 15/03/16.
 */
public class Contacto {

    private String nombre;
    private String telefono;
    private String email;
    private int foto;

    public Contacto(int foto, String nombre, String telefono, String email) {
        this.foto = foto;
        this.nombre = nombre;
        this.telefono = telefono;
        this.email = email;
    }

    public String getNombre() { return nombre; }

    public void setNombre(String nombre) { this.nombre = nombre; }

    public String getTelefono() { return telefono; }
}

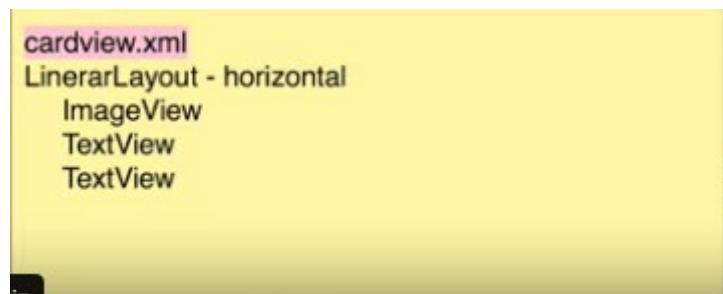
```

Y también voy a crear sus getters y setters.

Y vamos a darle por aquí un clic derecho. Me parece que si colocamos alt enter. Si, si colocamos alt enter aquí, finalmente de foto, puedo tener un acceso directo a create getter and setter de foto.

Y hasta ahora, ya he terminado pues este paso.

Ahora vamos a definir nuestro card view.

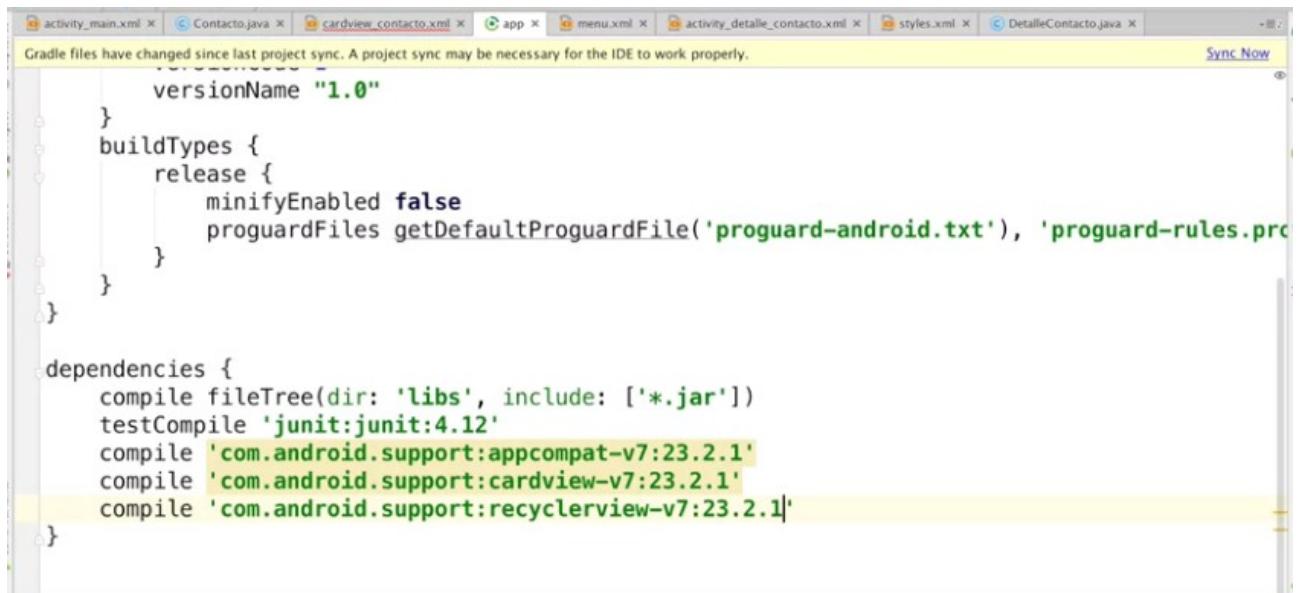


Aquí vamos a ponerle: cardview.xml. Esto será un nuevo layout. Un nuevo layout será un nuevo archivo que esté viviendo en nuestra carpeta layout.

Y que se va a llamar cardview.xml. A lo mejor podríamos colocarle algo un poco más amigable para el proyecto.

Podría ponerle cardview_contacto para que pueda identificar a qué corresponde el card_view.

Luego, para poder usar cardview y recyclerview, se debe editar el gradle.build

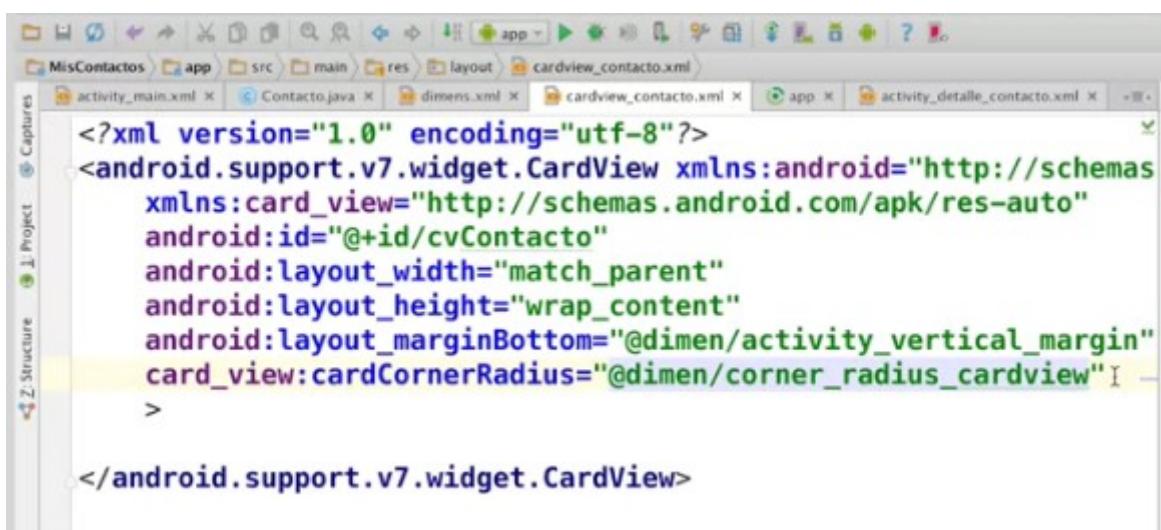


The screenshot shows the Android Studio interface with the build.gradle file open. The code defines a buildType named 'release' with minifyEnabled set to false and proguardFiles set to 'proguard-android.txt' and 'proguard-rules.pro'. It also lists dependencies for 'fileTree', 'junit:junit:4.12', and three versions of 'com.android.support' libraries: 'appcompat-v7:23.2.1', 'cardview-v7:23.2.1', and 'recyclerview-v7:23.2.1'.

```
versionName "1.0"
}
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.2.1'
    compile 'com.android.support:cardview-v7:23.2.1'
    compile 'com.android.support:recyclerview-v7:23.2.1'
}
```

Ahora se edita el cardview_contacto.xml



The screenshot shows the Android Studio interface with the cardview_contacto.xml file open. The XML code defines a CardView with various attributes: 'id' set to '@+id/cvContacto', 'layout_width' set to 'match_parent', 'layout_height' set to 'wrap_content', 'layout_marginBottom' set to '@dimen/activity_vertical_margin', and 'card_view:cardCornerRadius' set to '@dimen/corner_radius_cardview'.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res-auto"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/cvContacto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="@dimen/activity_vertical_margin"
    card_view:cardCornerRadius="@dimen/corner_radius_cardview" >

</android.support.v7.widget.CardView>
```

```
LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >

    <ImageView
        android:id="@+id/imgFoto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/shock_rave_icon"
        />

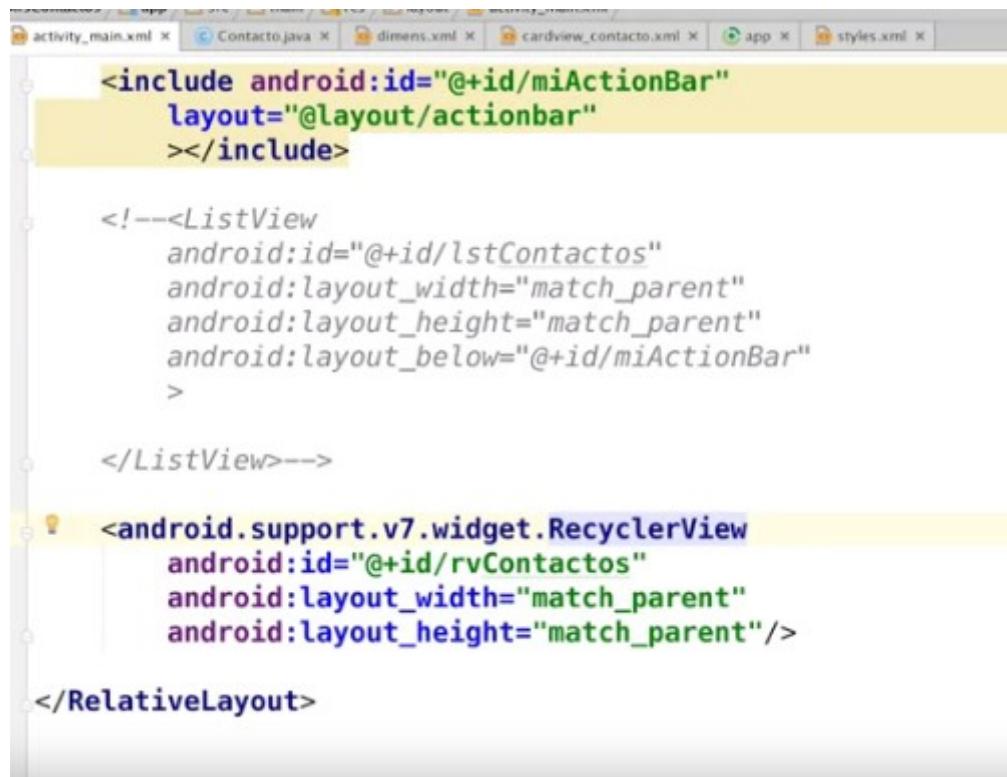
</LinearLayout>

<TextView
    android:id="@+id/tvNombreCV"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pnombre"
    android:textSize="@dimen/nombre_cardview"
    android:textStyle="bold"
    />

<TextView
    android:id="@+id/tvTelefonoCV"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/ptelefono"
    />

</LinearLayout>
```

Ahora en activity_main.xml



```

<include android:id="@+id/miActionBar"
    layout="@layout/actionbar"
></include>

<!--<ListView
    android:id="@+id/lstContactos"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@+id/miActionBar"
    >

</ListView>-->

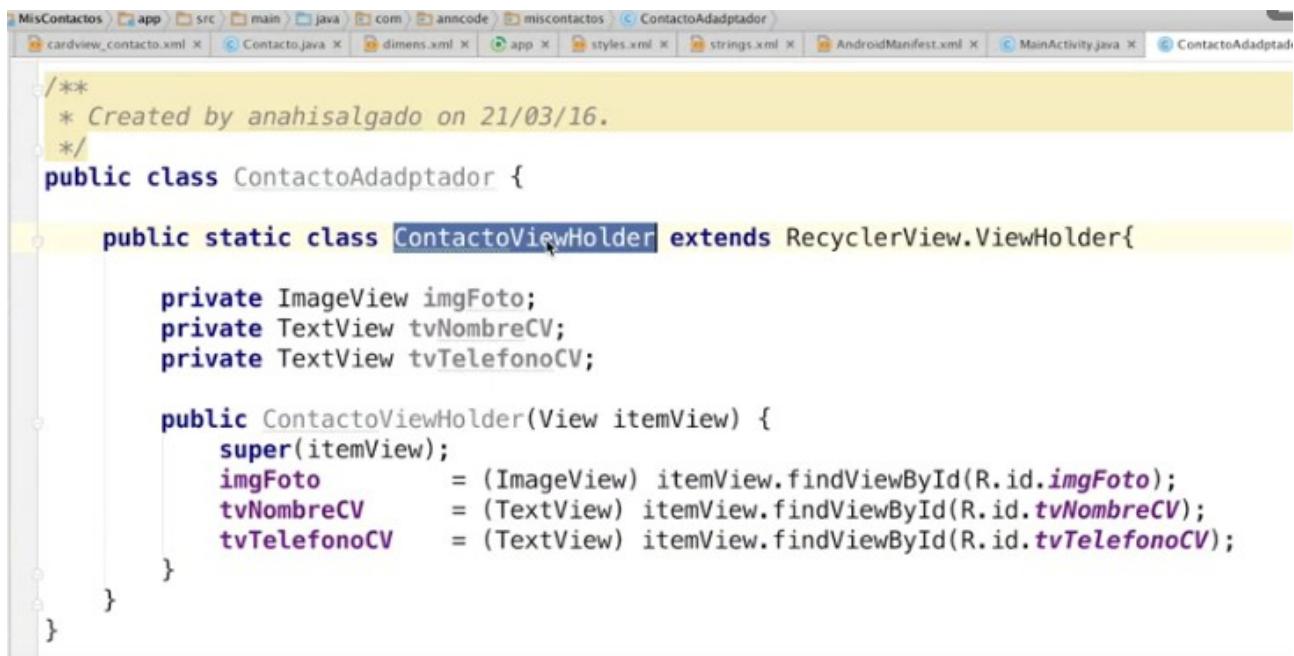
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.RecyclerView
    android:id="@+id/rvContactos"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>

</RelativeLayout>

```

Ahora, lo que se necesita es un adaptador para que interactúen el recyclerview y el cardview y que lo vaya llenando.

El adaptador se maneja en una clase nueva estática ViewHolder y ahí se declaran todos los views.



```

/*
 * Created by anahisalgado on 21/03/16.
 */
public class ContactoAdaptador {

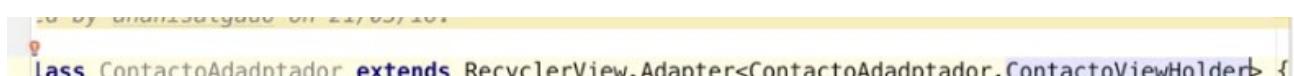
    public static class ContactoViewHolder extends RecyclerView.ViewHolder {

        private ImageView imgFoto;
        private TextView tvNombreCV;
        private TextView tvTelefonoCV;

        public ContactoViewHolder(View itemView) {
            super(itemView);
            imgFoto = (ImageView) itemView.findViewById(R.id.imgFoto);
            tvNombreCV = (TextView) itemView.findViewById(R.id.tvNombreCV);
            tvTelefonoCV = (TextView) itemView.findViewById(R.id.tvTelefonoCV);
        }
    }
}

```

Pero ahora queda pendiente que reciba las cosas el adaptador



```

lass ContactoAdaptador extends RecyclerView.Adapter<ContactoAdaptador.ContactoViewHolder> {

```

Ésto hace que obligatoriamente deba crear los 3 métodos:

```

ArrayList<Contacto> contactos;
@Override
public ContactoViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    return null;
}

@Override
public void onBindViewHolder(ContactoViewHolder holder, int position) {

}

@Override
public int getItemCount() { //Cantidad de elementos que contiene mi lista
    return contactos.size();
}

public static class ContactoViewHolder extends RecyclerView.ViewHolder{

```

En onCreateViewHolder:

```

View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.cardview_contacto)
    .layoutInflater.from(parent.getContext()).inflate(R.layout.cardview_contacto, parent, false);

```

En ésta linea es la que se indica cual layout es el que va a refrescar.

Al final queda

```

public ContactoViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.cardview_contacto)
        .layoutInflater.from(parent.getContext()).inflate(R.layout.cardview_contacto, parent, false);
    return new ContactoViewHolder(v);
}

```

Lo que hace es pasarle al constructor el view inflado para que le asigne las cosas definidas en el mismo.

Por último, en el onBindViewHolder es donde se van a setear cada uno de los valores y se invoca cada vez que se recorre la lista.

```

@Override
public void onBindViewHolder(ContactoViewHolder contactoViewHolder, int position) {
    Contacto contacto = contactos.get(position);
    contactoViewHolder.imgFoto.setImageResource(contacto.getFoto());
    contactoViewHolder.tvNombreCV.setText(contacto.getNombre());
    contactoViewHolder.tvTelefonoCV.setText(contacto.getTelefono());
}

```

Por último se hace el método constructor que construye la lista de contactos:

```

/**
 * Created by anahisalgado on 21/03/16.
 */
public class ContactoAdaptador extends RecyclerView.Adapter<ContactoAdaptador>

    public ContactoAdaptador(ArrayList<Contacto> contactos){
        this.contactos = contactos
    }

    ArrayList<Contacto> contactos;

```

Por último, en el .java correspondiente al activity que tenga el recyclerview:

```
public class MainActivity extends AppCompatActivity {

    ArrayList<Contacto> contactos;
    private RecyclerView listaContactos;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toolbar miActionBar = (Toolbar) findViewById(R.id.miActionBar);
        setSupportActionBar(miActionBar);

        listaContactos = (RecyclerView) findViewById(R.id.rvContactos);
```

Ahora, lo que falta es indicarle cómo quiero mostrar ese recyclerView, si lo quiero como lista por ejemplo:

```
LinearLayoutManager llm = new LinearLayoutManager(this);
llm.setOrientation(LinearLayoutManager.VERTICAL);

listaContactos.setLayoutManager(llm);
```

O sea que mi recyclerView se comporte como un LinearLayout:

Posteriormente se inicializa el adaptador:

```
public void inicializaAdaptador(){
    ContactoAdaptador adaptador = new ContactoAdaptador(contactos);
    listaContactos.setAdapter(adaptador);
}
```

RecyclerView onClick

Hagamos que los elementos de nuestro RecyclerView tengan asignado un método onClick, para que al seleccionar alguno nos lleve al detalle de cada contacto.

A continuación haremos que nuestros items sean clickables y nos lleven al detalle del contacto dando click en la imagen nos llevará a la actividad DetalleContacto.

Para esto iremos a nuestro archivo ContactoAdaptador y ubicaremos el método onBindViewHolder dentro de él colocaremos un onClickListener a la imagen, como se muestra:

```
@Override  
public void onBindViewHolder(ContactoViewHolder contactoViewHolder, int position) {  
    final Contacto contacto = contactos.get(position);  
    contactoViewHolder.imgFoto.setImageResource(contacto.getFoto());  
    contactoViewHolder.tvNombre.setText(contacto.getNombre());  
    contactoViewHolder.tvTelefono.setText(contacto.getTelefono());  
  
    contactoViewHolder.imgFoto.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            Toast.makeText(activity, contacto.getNombre(), Toast.LENGTH_SHORT).show();  
        }  
    });  
}
```

Hemos puesto un mensaje Toast, que nos muestra el nombre del contacto, al dar click en su foto. Como observas el mensaje Toast recibe un objeto activity como contexto este lo estaremos declarando como variable global en nuestra clase ContactoAdaptador y lo inicializaremos en el constructor como se observa:

```
ArrayList<Contacto> contactos;  
Activity activity;  
  
public ContactoAdaptador(ArrayList<Contacto> contactos, Activity activity) {  
    this.contactos = contactos;  
    this.activity = activity;  
}
```

Y este activity lo estaremos pasando cuando creemos una instancia de este adaptador en nuestro método inicializaAdaptador()

```
public ContactoAdaptador adaptador;  
private void inicializaAdaptador(){  
    adaptador = new ContactoAdaptador(contactos, this);  
    rvContactos.setAdapter(adaptador);  
}
```

Ahora solo nos faltará colocar nuestro intent para que nos lleve a la actividad DetalleContacto

```

@Override
public void onBindViewHolder(ContactoViewHolder contactoViewHolder, int position) {
    final Contacto contacto = contactos.get(position);
    contactoViewHolder.imgFoto.setImageResource(contacto.getFoto());
    contactoViewHolder.tvNombre.setText(contacto.getNombre());
    contactoViewHolder.tvTelefono.setText(contacto.getTelefono());

    contactoViewHolder.imgFoto.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(activity, contacto.getNombre(), Toast.LENGTH_SHORT).show();
            Intent intent = new Intent(activity, DetalleContacto.class);
            intent.putExtra("nombre", contacto.getNombre());
            intent.putExtra("telefono", contacto.getTelefono());
            intent.putExtra("email", contacto.getEmail());
            activity.startActivity(intent);
        }
    });
}

```

Enviamos los parámetros y los recibimos en el activity DetalleActivity.java

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detalle_contacto);

    Bundle parametros = getIntent().getExtras();
    String nombre = parametros.getString("nombre");
    String telefono = parametros.getString("telefono");
    String email = parametros.getString("email");

    tvNombre = (TextView) findViewById(R.id.tvNombre);
    tvTelefono = (TextView) findViewById(R.id.tvTelefono);
    tvEmail = (TextView) findViewById(R.id.tvEmail);

    tvNombre.setText(nombre);
    tvTelefono.setText(telefono);
    tvEmail.setText(email);
}

```

Botón de Like en RecyclerView

Agregaremos un botón de like a nuestro cardview para tener la posibilidad de dar like a un contacto en particular, Tan solo estaremos simulando esto, pero dejaremos listo el botón y su respectivo evento onClick.

Lo primero que haremos es editar nuestro archivo cardview_contacto.xml, agregaremos un view de tipo ImageButton, este tendrá sus atributos obligatorios y un id llamado btnLike además le colocaremos en el atributo android:background la imagen de <http://ic8.link/6227>

El código del layout cardview_contacto.xml se visualiza dela siguiente manera:

Y de la misma forma como lo hicimos con la imagen, en el método **onBindViewHolder** agregaremos el **onClickListener** para el botón como se muestra en el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/tvContacto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="@dimen/activity_vertical_margin"
    card_view:cardCornerRadius="@dimen/corner_radius_cardview">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="@dimen/activity_horizontal_margin"
    >

        <ImageView
            android:id="@+id/imgFoto"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:src="@drawable/shock_rave_icon"
        />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
        >
            <ImageButton
                android:id="@+id/btnLike"
                android:layout_width="38dp"
                android:layout_height="38dp"
                android:background="@mipmap/ic_like"
            />
    
```

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_marginLeft="@dimen/left"
    >
    <TextView
        android:id="@+id/tvNombreCV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pnombre"
        android:textSize="@dimen/nombre_cardview"
        />

    <TextView
        android:id="@+id/tvTelefonoCV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ptelefono"
```

En nuestra clase **ContactoViewHolder** agregaremos el view como lo hicimos con los otros:

```
public static class ContactoViewHolder extends RecyclerView.ViewHolder {

    private ImageView imgFoto;
    private TextView tvNombre;
    private TextView tvTelefono;
    private ImageButton btnLike;

    public ContactoViewHolder(View itemView) {
        super(itemView);

        imgFoto = (ImageView) itemView.findViewById(R.id.imgFoto);
        tvNombre = (TextView) itemView.findViewById(R.id.tvNombreCV);
        tvTelefono = (TextView) itemView.findViewById(R.id.tvTelefonocV);
        btnLike = (ImageButton) itemView.findViewById(R.id.btnLike);
    }
}
```

Y de la misma forma como lo hicimos con la imagen, en el método onBindViewHolder agregaremos el onClickListener para el botón como se muestra en el siguiente código:

```
contactoViewHolder.btnAddLike.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Toast.makeText(activity, "Diste like a " + contacto.getNombre(),  
                Toast.LENGTH_SHORT).show();  
    }  
});
```

Por el momento solo colocamos un Toast que nos indica a qué contacto le dimos like.

Añadiendo un App Bar Material Design y Ejemplo

Otro de los elementos que tenemos también disponibles con material design, es nuestro appbar o nuestra barra de navegación que la vas a encontrar en la parte superior de las aplicaciones.

Esta barra de navegación se ha incluido como un elemento nuevo de material design. Dado que esta va a tener una elevación por sobre de algunos de los elementos que van a componer tu interfaz gráfica.

Entonces, si observamos un poco en nuestro proyecto, tengo aquí el layout donde está mi lista, mi list view. Recuerda que es nuestro list view con nuestros contactos.

Solamente están los nombres de los contactos y ahora el preview que estoy teniendo para esta lista de contactos es el API 23. Si yo cambio para el API 22, bueno no sucede mucho, solamente un problema en el rendereo de la información, pero no.

Se sigue viendo mi app bar, si me bajo un poco más al API 21, en ese momento el app bar ya no está disponible.

Si voy al API 19, el app bar tampoco está disponible, y si voy un poco más abajo, tengo un app bar que ni siquiera tiene que ver con el estilo o con el diseño que yo había definido en mis colores.

De hecho es gris y tiene otro estilo, otro diseño.

Básicamente lo que se planea en este módulo es que nosotros podamos diseñar nuestra propia app bar. Y que podamos unificarla que sea igual para versiones de material design en adelante, y versiones anteriores a Android Lollipop. Que nuestra barra de navegación que está en la parte superior se vea igual en cualquier versión de Android.

Bien, entonces para ello. Lo primero que tengo que hacer es ir a mi archivo styles.

El archivo esta vez lo voy a ubicar en la carpeta Res, en la carpeta Values, y ahí está mi archivo styles.

Y lo primero que detecto es que el tema con el cual se ha creado esta aplicación, se llama Theme App Compat Light, de eso no hay problema.

Es uno de los temas de compatibilidad de material design.

Pero observamos que dice Dark Action Bar. Es decir, por default nos está colocando el tema, una Action Bar, una barra.

Que esta barra va ser variable o va ser con estilos diferentes dependiendo de la versión en la que yo esté en este momento visualizando mi aplicación.

Entonces lo que voy hacer primeramente, es eliminar esa barra, que me está proveyendo el tema. La voy a eliminar y lo haré colocando el tema theme AppCompat.Light y NoActionBar. Ésta será la extensión que necesito, NoActionBar.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

No necesitamos una action bar que nos prevea el tema porque yo voy a diseñar mi propia action bar.

Ustedes y yo diseñaremos nuestra propia action bar. Entonces como observas, pues en el API 15 ya no está presente esa barra que teníamos anteriormente. Si nos vamos a otra API, pues tampoco está presente. Y esa es precisamente la idea. La idea era quitarla, eliminarla, deshacernos de ella.

Para generar una action bar más linda y con diseño de material design.

Entonces para desarrollar nuestra action bar, lo que vamos a tener que hacer es crear un nuevo layout. Un nuevo layout donde ahí vamos a diseñar cómo queremos que se vea nuestra app bar.

Para crear un nuevo layout, voy a dar click derecho en layout, new layout resource file, y ahí voy a escribir el nombre de mi action bar. Action bar, actionBar, tal cual.

Esto por default pues bueno, aquí estoy definiendo que me cree un linear layout, o lo que sea, finalmente pues voy a eliminar todo lo que esté por ahí.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</LinearLayout>
```

Bien, tengo esto, es un layout en blanco, se llama action bar, y voy a ir a la vista de texto.

Lo primero que tengo en la vista de texto es mi linear layout que me crea por default. Vamos a quitarlo de ahí, vamos a borrarlo.

Este linear layout no va a servir para crear una interfase como lo hemos estado haciendo, sino este layout lo único que va a ser es definir nuestra barra, es para lo único que va a servir.

Solamente ahí tendremos como widget nuestra tool bar, nuestra barra. Y bueno, este widget lo vamos a tener disponible en android.support.v7.widget.Toolbar. Éste es el widget que buscamos.

Lo selecciono y cierro. Cierro la etiqueta.

Ahora, aquí yo necesito colocar mis atributos obligatorios para cada etiqueta. Voy a colocar android: layout width, ¿y qué observamos? Observamos que no me está reconociendo el atributo Android, esto es porque no está dado de alta el name space que define ese atributo. Entonces si observas por ahí, lo único que hice fue dar control, barra espaciadora, control barra espaciadora. Y me salió esa sugerencia, que necesito importar el name space para esta propiedad.

Entonces voy hacer alt enter, y automáticamente ya me ha traído el name space que necesito. Nada más voy a quitar esta propiedad Android, voy a pasar este name space para arriba, y dentro del pico paréntesis voy a estar escribiendo mis propiedades.

Entonces voy a colocar Android y voy a empezar con mis comandos mis propiedades obligatorias. Voy a colocar en el width, el width de la barra debe ser match parent, siempre debe ajustarse al padre, al tamaño de la ventana de la aplicación.

Entonces vamos a colocar match parent. En este momento no nos lo sugiere, porque no detecta que en algún momento ya hayamos escrito o que you esté presente esa palabra en el editor.

Ahora vamos con nuestro height. Y para el height vamos a utilizar un recurso y lo vamos a traer. Vamos a poner attr, diagonal, action, bar, size. Esto lo que hace es que trae un atributo donde me define el action bar size.

En este momento el color de la barra es transparente o es color blanco, por eso no se observan, no se identifican dónde está. Vamos precisamente a definir eso. Vamos a definir el color de fondo de nuestra barra, voy a colocar Android background. Y el color de una barra de estado siempre debe ser el primary color. Nuestro color primary.

Ahí ya se está observando.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="@color/colorPrimary"
>

</android.support.v7.widget.Toolbar>
```

Y nuestro dark primary color está en la parte superior.

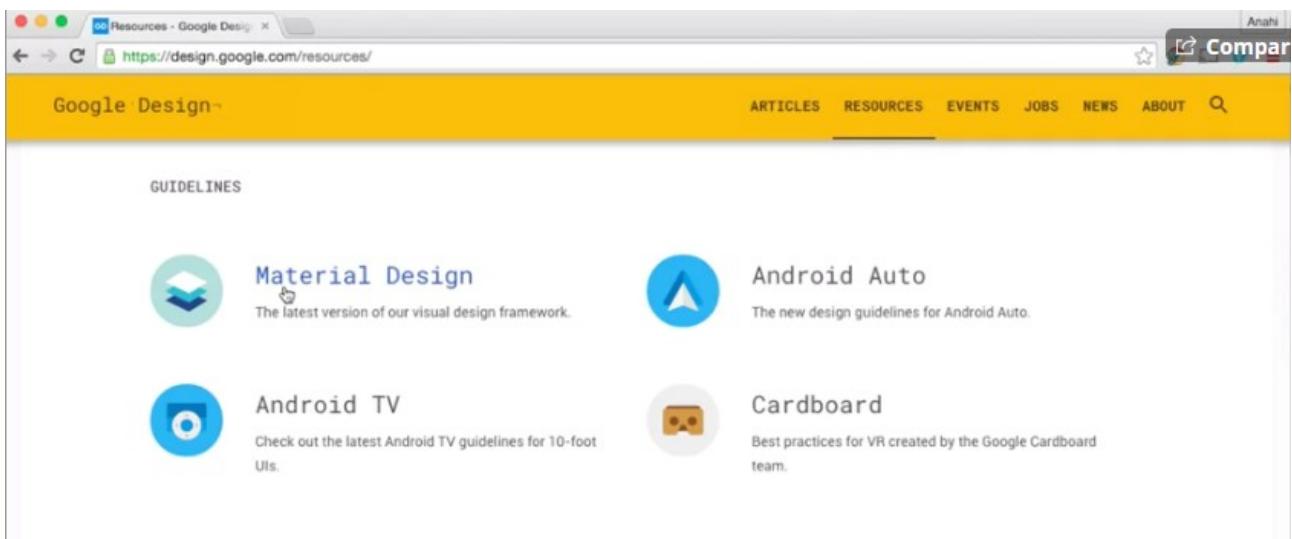
Que es nuestra barra de notificaciones. Bien, ahí está nuestro color primary.

Y otra cosa que también debemos colocar para nuestra barra es la elevación. Hay un punto muy interesante que no hemos tocado aun, y es que todos los elementos de material design. Todos y cada uno tienen asignada una elevación. Una elevación que debe tener un elemento con respecto a otro. Recuerda que con material design ya no tenemos solamente un eje X y un eje Y para diseñar. Sino ahora tenemos también un eje Z.

Un eje Z que va a significar que nuestro teléfono va estar en esta forma. Nuestros elementos así, y nuestros elementos van a estar flotando en el eje Z.

Para poder dar una coordinación a nuestros elementos en qué elemento va a estar arriba de otro, cuál va a estar debajo. Google a colocado una guía. En Design.Google.com o Google.com/design.

Si vamos a la pestaña de resources. Bajamos un poco, y vamos aquí donde dice guide lines material design. Vamos a entrar. Y tenemos que buscar aquí en el menú la parte de estilo.



Tenemos que buscar. En el menú. Donde dice "what is material?". "Elevation and shadows". Vamos a dar click ahí. Elevation.

A screenshot of a web browser displaying the 'Introduction - Material design' page from Google's design specifications. The left sidebar has a menu with 'Material design' selected, and 'Introduction' is the active sub-item. The main content area has a teal header with the word 'Introduction'. Below it, there's a large image of a colorful geometric pattern. To the left of the main content, there's a sidebar with sections like 'What is material?', 'Environment', 'Material properties', 'Elevation and shadows', 'Animation', 'Style', 'Layout', and 'Components'. The main content area has two columns. The left column is titled 'Bold, graphic, intentional' and discusses the foundational elements of print-based design. The right column is titled 'Motion provides meaning' and discusses motion's role in reinforcing the user as the prime mover. At the bottom of the main content, there's a footer with a copyright notice.

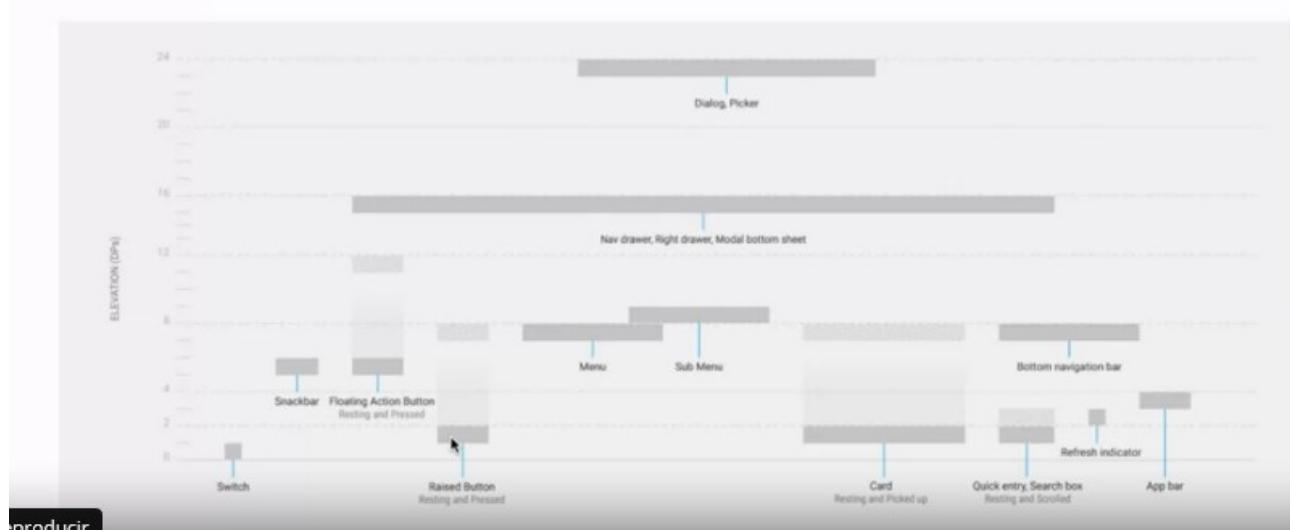
Y ahora, aquí me muestra Las medidas de elevación que debe tener cada elemento. Por ejemplo, recuerda que todo está en las unidades de medida que son los dps. El componente dialog picker debe estar en una elevación de 24 dps. Un navigation drawer debe estar en una elevación de 16 dp.

Mucho más abajo. Un right drawer, también un modal button sheet. Y tenemos un menú. Un Snackbar qué elevación puede tener.

El Appbar que es lo que estamos nosotros diseñando. Nuestra Appbar debe estar en una elevación de 4 dps.

A lo mejor ésto es un poco confuso.

Más abajo tenemos aquí, en el gráfico, una forma de estar viendo mejor nuestras elevaciones.



Como vemos el switch está en una elevación de 1dp. Un raise button está en una elevación de 2dp.

Un card también está en una elevación de 2. Tenemos un refresh indicator, está en una elevación de tres. El appbar elevación de cuatro. Un Snackbar en una elevación de seis y vemos que el navigation drawer está en la 16, y dialog o los picker están hasta arriba.

Esos siempre debe estar en la parte superior en 24dp.

Entonces aquí podemos ver la elevación que tiene cada elemento, írsele asignando para tener una mejor coordinación con todos mis elementos.

Bien, entonces vemos aquí que hemos creado un app bar, pero el app bar no tiene elevación. Simplemente se ve plano, el color se ve plano.

Entonces, el app bar debe tener una elevación de 4dp. Vamos a dar de alta en el archivo dimens, nuestra elevación del app.

Vamos a poner elevación action bar, y dijimos que debe ser 4dp.

```
version="1.0" encoding="utf-8"?>
<id.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/r
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="@color/colorPrimary"
    android:elevation="@dimen/elevacion_actinbar" ->

</id.support.v7.widget.Toolbar>
```

Ahora si regresamos a nuestro layout y colocaremos nuestro recursos Android, propiedad Android elevation. Y colocaremos elevation action bar.

Y observamos que ya se ha pintado una sombra. Una sombra que nos está indicando la elevación de ese elemento. Si yo coloco de pronto 10dp, la sombra se incrementa. Entre mayor sombra tenga en un elemento la elevación es mucho mayor.

Ojo, no hay que abusar de estas elevaciones. Es muy recomendable seguir las métricas de cómo se nos están indicando las elevaciones para cada elemento.

Para este elemento está definido con 4dp. Con esto sería todo para declarar nuestro toolbar y ahora vamos a proceder simplemente a insertarlo en nuestro layout.

Tenemos este toolbar como un recurso adicional, como un recurso independiente. Entonces yo iré a mi activity main, en la vista de texto tengo que mi activity name, recuerda éste tiene la lista. La lista de contactos.

Antes de mi text view, antes de mi list view, voy a colocar. Mi toolbar. Con la etiqueta include, puedo incluir un layout o puedo incluir un recurso de XML en este layout.

Entonces colocaré Android id, le voy a colocar id, y le voy a poner miActionBar. Y con la etiqueta, con la propiedad layout, voy a definir a qué layout corresponde éste. Coloco el que acabamos de crear.

Y aquí está, aquí está nuestro actionBar, y puedo incrustarlo en todos los layout que necesite.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.anncode.miscontactos.MainActivity">

    <include android:id="@+id/miActionBar"
        layout="@layout/actionbar"
        ></include>

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

    </ListView>

</RelativeLayout>
```

Bien, con esto ya hemos creado un actionbar. Si nosotros corremos esto. En nuestro teléfono. La verdad es que si lo corremos en un teléfono. Si lo corremos en un teléfono con sistema operativo marshmallow o lollipop, la verdad es que esto va a ser casi invisible, pero se alcanza a ver el action bar. Se ve ahora diferente.

Como vemos aquí en nuestro layout, está tapando el primer elemento de la lista, así que voy a definir por acá la posición del list view. Despues vamos a ver cómo coordinar todos estos elementos para que puedan quedar sobrepuertos uno encima de otro y lo haremos con otro tipo de layout que lo veremos más adelante.

Simplemente a esta lista voy a decirle que esté debajo de mi action bar. Que siempre se mantenga ahí debajo.

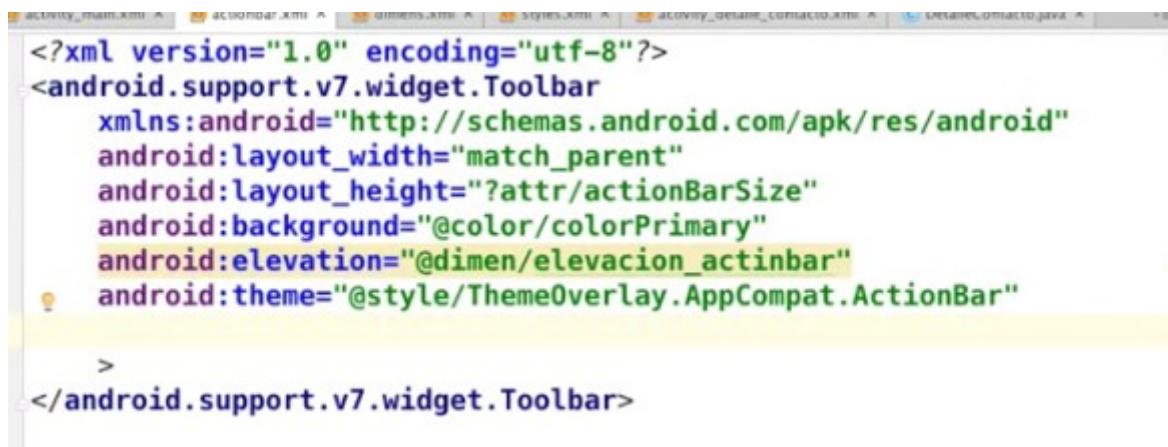
```
<include android:id="@+id/miActionBar"
        layout="@layout/actionbar"
        ></include>

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

Y ahí está, ahora si están todos nuestros elementos. Si yo entro a un elemento aquí veo mi action bar, de hecho se alcanza a ver una sombra que es la que define la posición, que es la que define la elevación.

De esta forma podemos crear un action bar, un action bar que sea compatible con versiones anteriores a lollipop y con versiones también superiores a lollipop.

Otra cosa que también podemos colocar en nuestro tool bar en nuestro action bar es la propiedad Android theme. Que esto lo que nos ayudará es a que, bueno, nuestra app bar. Compact action bar, nuestra app bar tenga también por default el estilo de material design. Digamos que esto es a prueba de todo.

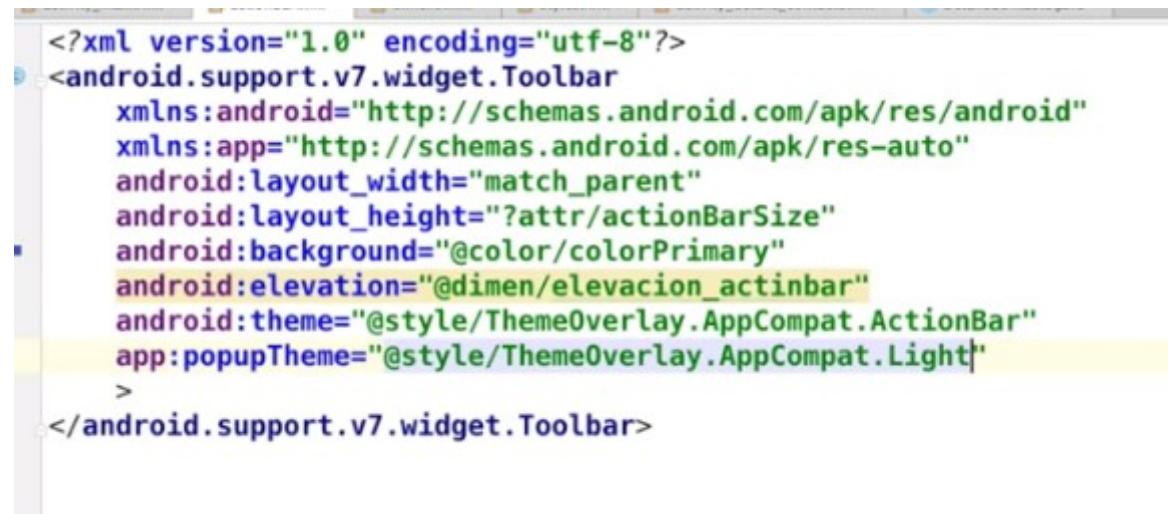


```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="@color/colorPrimary"
    android:elevation="@dimen/elevacion_actinbar"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar">
</android.support.v7.widget.Toolbar>
```

Colocaremos también una propiedad que es app que no está declarada y que necesito declarar también su namespace. Cuando escribo app, automáticamente ya me está sugiriendo que puedo colocar el name space para esa propiedad.

Si no te sale el name space, por acá, simplemente puedes dar Alt enter y automáticamente se va a importar el name space para que la propiedad app funcione.

Voy a dar app, dos puntos, popup theme, colocaré @style theme overlay, que es lo mismo que está arriba, AppCompat.Light.

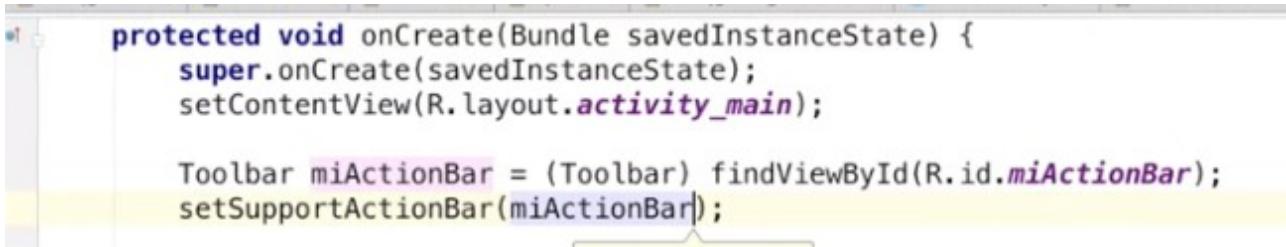


```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="@color/colorPrimary"
    android:elevation="@dimen/elevacion_actinbar"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light">
</android.support.v7.widget.Toolbar>
```

Esto es dependiendo del tipo de tema que hayas definido. Bueno con estos dos estamos diciendo que automáticamente este elemento tome todo el estilo de material design.

Y un último paso que debemos realizar es en nuestro main activity y en cada actividad en donde estemos trabajando nuestro app bar, debemos también declararlo.

Lo voy a declarar inmediatamente después de set content view. Voy a colocar Toolbar, tenemos aquí dos opciones.



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

    Toolbar miActionBar = (Toolbar) findViewById(R.id.miActionBar);
    setSupportActionBar(miActionBar);
```

Colocaré el toolbar support V7, toolbar y colocaré mi action bar. Y colocaré toolbar y pondré findViewById(R.id.miActionBar), que es la que declaré por allá. Tengo que colocar set support action bar, para completar el soporte para que esto se vea muy bien en todas mis pantallas.

Ahora si puedo correrlo, puedo correr la aplicación puedo ponerlo en mi teléfono, y empezar a visualizar

Ahora es de este color, pero podemos modificarlo.

Igualmente en la pantalla de detalle de contacto, no aparece el texto porque no he colocado el objeto toolbar como debería de ser. También en mi clase detalle contacto.

Y muy bien, es bien importante tener esto en nuestros proyectos. Poder manejar toolbars o actionbars, personalizadas para que bueno, puedas darle toda la personalidad que quieras lograr a tu aplicación. Y que además sea compatible con todas las versiones de Android, y tenga el elemento de material design.

Navegación hacia atrás

para qué y cómo

Es muy importante que notemos una diferencia entre el nombramiento de nuestros botones.

El botón de atrás y el botón de subir.

El botón de subir va a ser el botón que va a estar localizado en tu aplicación móvil en la parte superior, en tu app bar.

Y el botón de subir se va a estar comportando de una forma jerárquica. Una forma jerárquica conforme vas navegando dentro de tus activities.

Mientras que el botón de back, es utilizado para navegar en el orden inverso en que estuviste navegando en tus aplicaciones de una forma cronológica.

Todas las pantallas que manejes en tu aplicación móvil, deben tener el botón de subir o el botón de regreso que estará ubicado en la parte superior de nuestro action bar.

Todas estas pantallas a excepción de la pantalla principal o la primera pantalla que será nuestro main activity.

Hacer esto es muy sencillo, simplemente tenemos que indicar, como se está observando en este código.

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".SegundoActivity"
    android:parentActivityName=".MainActivity">

    <!-- Soporte para versiones 4.0 o menores -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
</application>

</manifest>
```

Tenemos que indicar en nuestro Android manifest, quién será la actividad padre de la actividad hijo en la que estás ubicado, que quiere regresar a una actividad anterior.

Además también debemos indicar en la actividad padre en nuestra clase de la actividad padre, dentro de nuestro método on create.

Debemos indicar con el método setDisplayHomeAsUpEnabled.

Con esto estamos indicando que estamos activando el uso de la navegación hacia atrás en nuestro action bar.

```
public class SegundoActivity extends AppCompatActivity {

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

        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    }
}
```

Y además estamos indicando que esta actividad será la actividad padre.

Ejemplo: Navegación hacia atrás

Bien, vamos a ver cómo podemos implementar un botón de regreso en nuestro proyecto.

Perfecto, yo tengo a continuación aquí, en mi Android Studio, ya he creado un proyecto.

Un proyecto que simplemente contiene dos pantallas. Una pantalla es un main activity, que aquí está. Y que tiene un botón que dice ir al segundo activity.

Adicionalmente también creé un segundo activity que se llama activity segundo. Y que únicamente contiene el texto activity 2.

Entonces lo que vamos a hacer es que vamos a colocar nuestro botón de regreso, vamos a habilitarlo en realidad.

Vamos a hacer que aparezca en nuestra barra de navegación y que cuando le den clic ahí pues debe regresar a nuestra otra pantalla.

Lo primero que tenemos que hacer es detectar cuál es la jerarquía de nuestras pantallas.

Yo hasta el momento tengo dos. Una actividad primera, la principal, y una segunda actividad.

Entonces la actividad primera, nuestro main activity, será por el momento mi actividad padre.

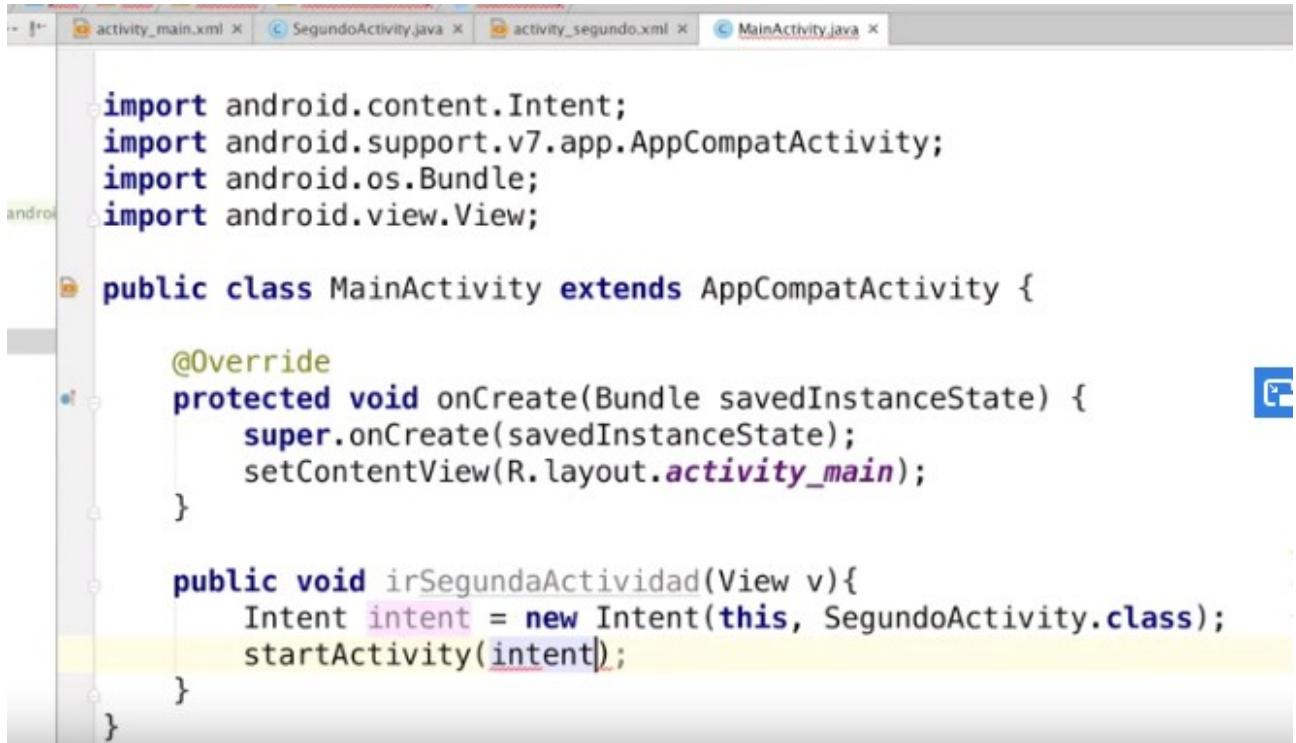
Eso es lo que tengo que identificar, la jerarquía. ¿Quién es mi actividad padre? Y posteriormente, ¿quién es mi actividad hijo? Entonces mi actividad padre es main activity y mi actividad hijo es segundo activity.

Entonces yo tengo aquí un botón, que simplemente vamos a poner por aquí un método public void.

Y vamos a ponerle ir a segunda actividad. Ir a la segunda actividad. Ese botón va a responder a éste método.

Y recuerda que para ir de una actividad a otra lo hacemos con un intent. Colocamos así en dónde estoy ubicada y a dónde quiero ir. Quiero ir a la segunda actividad.class.com.

Y finalmente disparo start activity, disparo el intent.



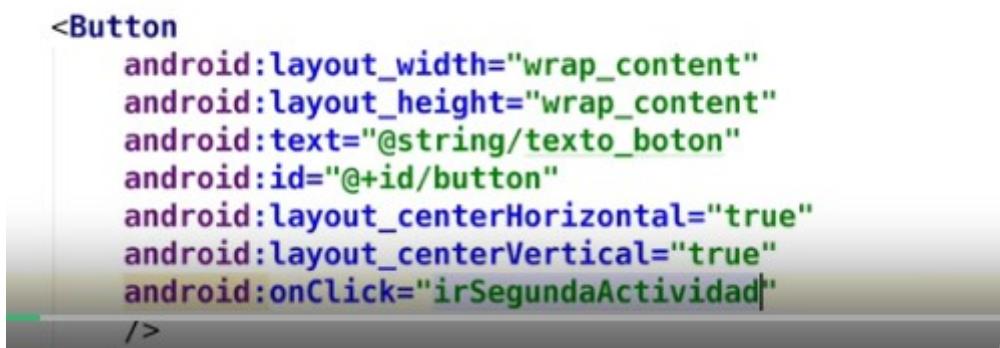
```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

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

    public void irSegundaActividad(View v){
        Intent intent = new Intent(this, SegundoActivity.class);
        startActivity(intent);
    }
}
```

Bien, con esto ya estoy yendo de una actividad a otra. Ahora nada más este método lo vamos a colocar en nuestro botón. Nuestro botón que yo ya di de alta por aquí.



Le voy a colocar por aquí Android on click, y aquí está, ir a la segunda actividad.

Si veo la vista de diseño, bueno, aquí está mi botón. Mi botón ya se ha acomodado, está centrado verticalmente y centrado horizontalmente.

Cuando lo presionas nos llevará a la segunda actividad. Aquí está mi segundo layout, este es el segundo layout que tengo.

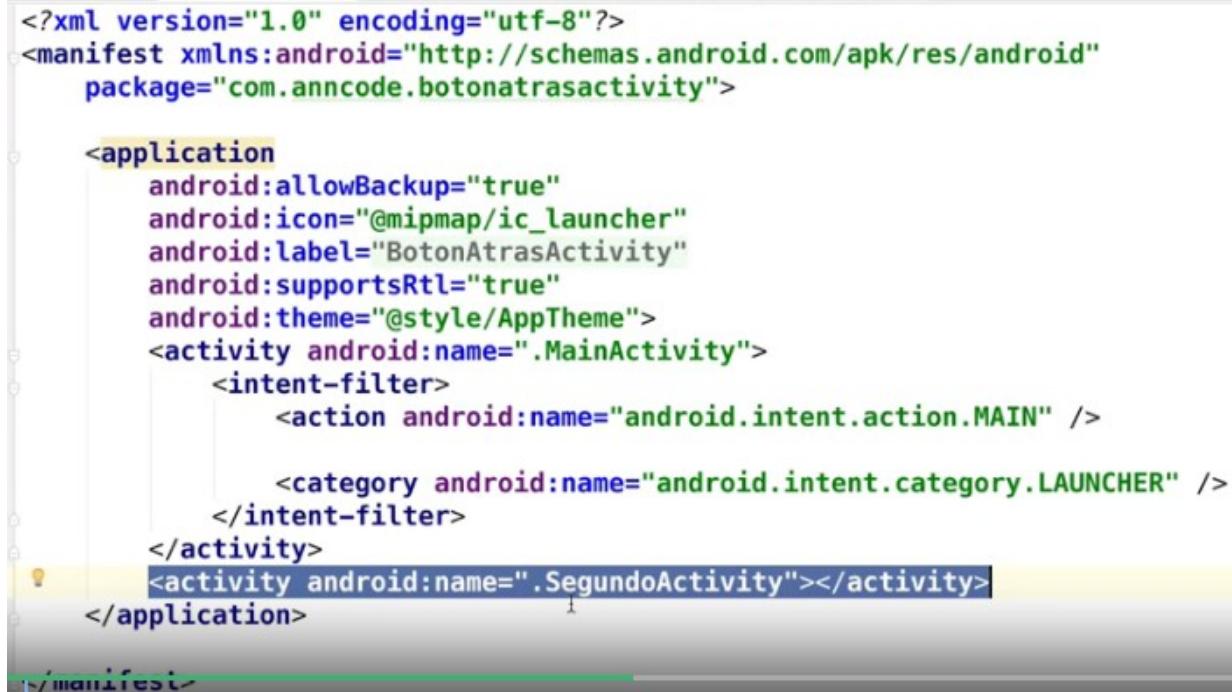
Entonces si probamos esto, vamos a correr esto. Lo había pasado. Entonces si corremos esto, vamos a esperar a que se construya.

De preferencia les digo a ustedes los editores que cuando esté tardando así como demasiado en correr, si se puede cortar.

Bueno, ya hemos corrido nuestra aplicación. Tenemos ésto, ir al segundo activity y finalmente nos lleva al segundo activity.

No tengo forma de regresarme a la actividad anterior a menos que presione el botón de regreso.

Entonces voy a habilitar esto precisamente. Lo primero que tengo que hacer es ir a mi archivo android manifest. Como lo hablamos hace un momento, tengo que ir a mi archivo android manifest.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.anncode.botonatrasactivity">

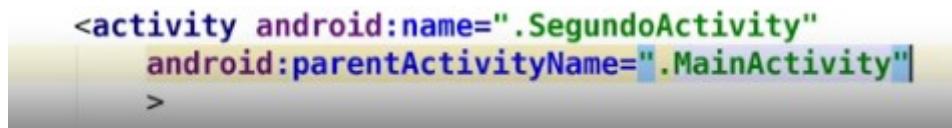
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="BotonAtrasActivity"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SegundoActivity"></activity>
    </application>

</manifest>
```

Y en mi archivo android manifest, tengo que ubicar la actividad hijo. Tengo que ubicar la actividad hijo, en este caso es segundo activity. Segundo activity, ahí debo colocar la propiedad android parent activity name. Android parent activity name, esa propiedad.

Y entonces ahora me va a decir quién es la actividad padre de segundo activity. La actividad padre de segundo activity es main activity.



```
<activity android:name=".SegundoActivity"
    android:parentActivityName=".MainActivity"
    >
```

Bien, selecciono eso y listo. Es todo lo que tengo que poner.

Ahora, para poder habilitar un soporte para versiones anteriores a Android 4.0 puedo colocar la etiqueta metadata. Entonces aquí simplemente te voy a poner un comentario que diga esto es para soporte para versiones 4.0 o menores.

Entonces ese soporte lo colocaré con el atributo android name, y entonces aquí colocaré android.support. Probablemente el IDE no te sugiera lo que estamos buscando pero básicamente hay que colocar con mayúsculas parent_activity. Y consecutivamente coloco también android:value.

Y coloco aquí quién es la actividad padre, quién es la actividad padre de segundo activity.

Si observas pues activity main se queda exactamente igual.

Esta es la etiqueta que configura main activity y esta es la etiqueta que configura segundo activity.

```
<activity android:name=".SegundoActivity"
    android:parentActivityName=".MainActivity">

    <!-- Soporte para versiones 4.0 o menores -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"%
    ></meta-data>
</activity>
```

Con esto hasta ahora solamente hemos configurado el flujo que va a seguir. Si tú tuvieras por acá otra actividad, bueno también tenemos que colocar, igualmente la jerarquía, la lógica de la jerarquía que va tener esa tercera actividad, quién será su padre.

Entonces vamos ahora a nuestro segundo activity. En nuestro segundo activity necesitamos que en la vista aparezca nuestro botón de regresar. Como observas pues aquí ya se ve algo. Y entonces tenemos que habilitar precisamente que ésto funcione, y que ésto funcione correctamente.

Entonces en segundo activity, en su método on create, voy a colocar getSupportActionBarsetDisplayHomeAsUpEnabled. Y le voy a colocar true.

```
package com.anncode.botonatrasactivity;

import ...

public class SegundoActivity extends AppCompatActivity {

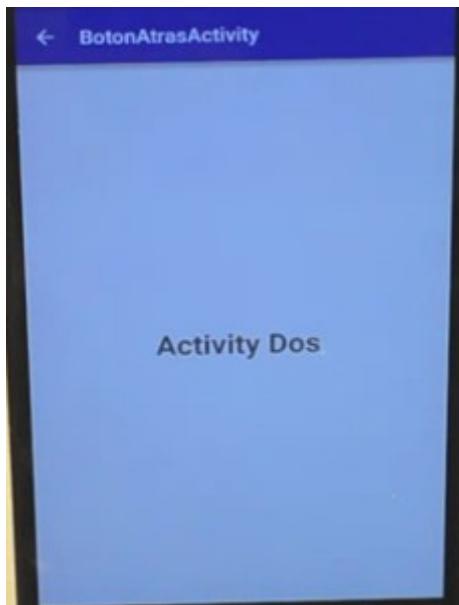
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_segundo);

        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    }
}
```

Ya se está viendo aquí nuestro botón de atrás. Pero con esto realmente nos aseguramos que el soporte y que todo lo que hemos configurado funcione adecuadamente.

Es todo lo que tenemos que hacer, solamente son estos dos archivos que tenemos que alterar.

Primeramente nuestro archivo android manifest. Y consecuentemente nuestra actividad hijo debemos colocar a dónde debe dirigirse, o más bien, en nuestra actividad hijo debemos habilitar el soporte para que todo ésto funcione adecuadamente. Entonces lo corremos.



Como observas, en la actividad padre no tengo botón de atrás. Cuando doy ir al segundo activity tengo la actividad dos. Y si presiono el botón de atrás entonces va a ir a nuestra actividad anterior.

Recuerda que este botón que está en la parte superior es un botón de subir. Simplemente estamos subiendo en la jerarquía.

Tú puedes controlarlo y puedes estar definiendo qué actividad se considera arriba o una actividad padre de otra.

Y el botón de back simplemente lo que hará es un retroceso cronológico de cómo es que hemos ido transcurriendo en todas las ventanas de nuestra aplicación. No necesariamente el botón de regreso nos va a regresar a la actividad padre. Podremos también, como vimos en secciones anteriores, controlar el botón de retroceso hacia qué actividad queremos ir.

Pero su configuración por defecto, él simplemente va a ir en retroceso cronológico. Y el botón que tenemos en la parte superior, este botón, sí sube en la jerarquía indicada.

Bien, de esta forma es como estamos habilitando un botón de subida o un botón de retroceso en nuestras actividades.

Semana 4

Menú de opciones

Los menús son un componente muy importante también para tus aplicaciones. Sobre todo cuando estas de pronto se vuelven muy sofisticadas, con muchas acciones o muchas opciones.

Es muy importante que todas estas opciones no las satures en la vista o en la interfaz de usuario, de una forma que la experiencia de usuario se comience a perder.

Todas estas opciones las puedes llevar a un menú donde pueden estar mejor ubicados. Y entonces pues puedes dar una mejor experiencia de usuario con tus aplicaciones.

Como seguramente has notado, a partir de Android 3.0, Android Honeycomb, también hemos tenido una evolución en el caso de los menús. A partir de estas versiones el botón de menús, si lo has notado, ya no aparece en nuestros teléfonos.

En versiones anteriores teníamos disponible un botón de menús que estaba ubicado a un lado del botón de home. Este botón cuando lo presionabas desplegaba automáticamente todos los menús que teníamos.

Hoy en día este botón ha desaparecido. Dándonos la posibilidad de ahora manejar los menús en nuestra barra de acciones.

A pesar de todos estos cambios que han ocurrido, la verdad es que la semántica y el API que tenemos para menús siguen siendo las mismas. Así que no te preocupes, podemos seguir utilizando todas las librerías y recursos para definir menús.

Es muy sencillo construir el menú.

Se puede construir desde código Java, solamente utilizando código Java, o también utilizando código XML. Es decir, utilizando un menú como recurso.

En Android tenemos tres tipos de menús.

- el menú de opciones.
- menús de contexto.
- menús pop-up.

El menú de opciones será nuestro menú principal en la aplicación. Todas las acciones principales o todas las acciones que quieras que el usuario las encuentre rápidamente deberán estar incluidas en este menú. Pueden ser configuraciones en tu aplicación, acciones de búsqueda, la redacción de un correo o la redacción de alguna otra cosa. Así que todas las opciones principales de tu aplicación tienen que estar construidas en un menú de opciones.

Ejemplo de menú de opciones

Voy a comenzar creando un menú de opciones.

Sabemos que el menú de opciones es aquél que aparece en la parte superior de nuestra barra, de nuestro ActionBar.

Entonces, vamos a comenzar por ahí.

Primeramente tenemos que decir que un menú, en términos del proyecto, de Android, un menú también se considera un recurso.

Tenemos dos formas de crear menús en Android, es a través de recursos XML y la otra es también a través de código Java, solamente código Java.

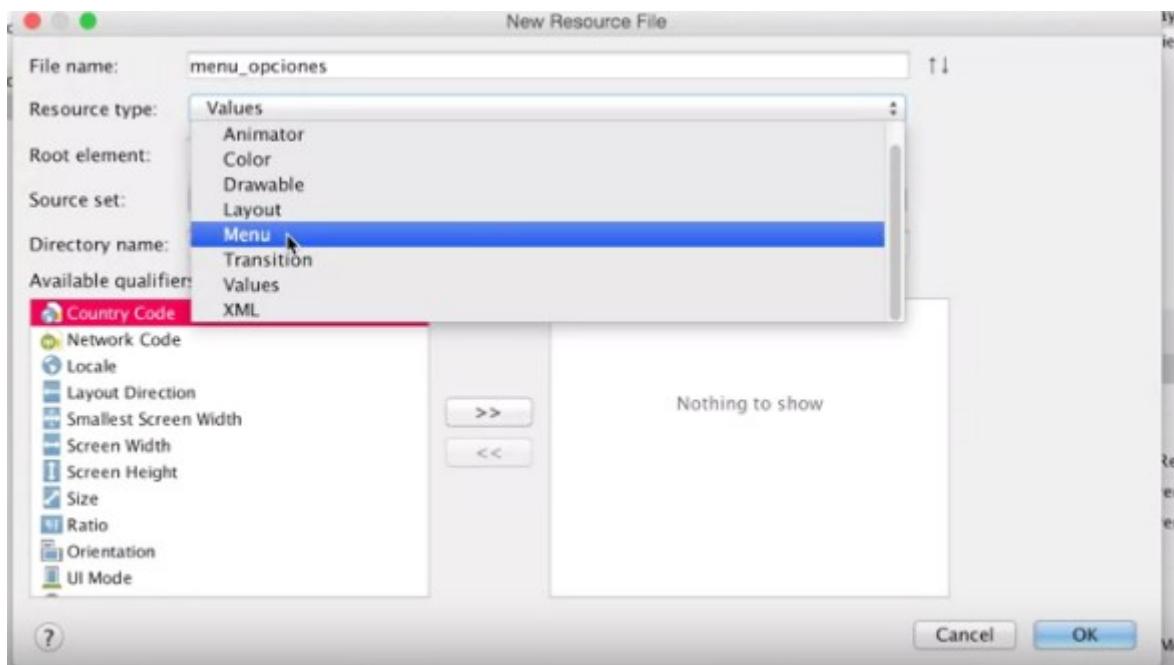
La forma como vamos estar trabajando en este momento es a través de un recurso xml.

Entonces, lo primero que tengo que hacer es precisamente crear ese recurso. Entonces, estoy situada en la carpeta res, bueno, antes de esto, simplemente debo decir que tengo un proyecto, he creado un proyecto de Android, un proyecto que se llama MenuOpcionesContextoPopup y lo único que tengo es nuestro Activity por Default y you, es todo lo que tenemos.

Entonces, creamos nuestro menú, nuestro menú como recurso. Voy a darle clic derecho en res, New y vamos a colocar Android resource file.

En File name, podemos colocar menú de opciones y en Values aquí vamos a colocar precisamente qué tipo de recurso es el menú que estamos creando o el archivo que estamos creando.

Precisamente en la opción de Resource type vamos a colocar que es un tipo Menú y vamos a dar OK.



Entonces, a continuación, you se creó nuestro recurso, simplemente, igualmente es una etiqueta xml y, bueno, podemos comenzar a añadir menús a través de la etiqueta item.

Voy a colocar aquí item y cerramos nuestra etiqueta.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item></item>
</menu>
```

Bien. He dejado un espacio porque voy a comenzar a escribir propiedades a este item.

Lo primero que voy a ponerle a este item es un Android Id.

Vamos a hacer un menú de opciones. Únicamente va a tener dos opciones, a lo mejor, podemos decir un menú de About o de Acerca de, de nosotros o acerca de esta aplicación y otro menú, podemos definirlo como un menú de Settings, un menú de configuraciones de la aplicación.

Entonces, puedo colocar aquí y puedo poner mAbout y ahorita vamos a continuar escribiendo cosas ahí.

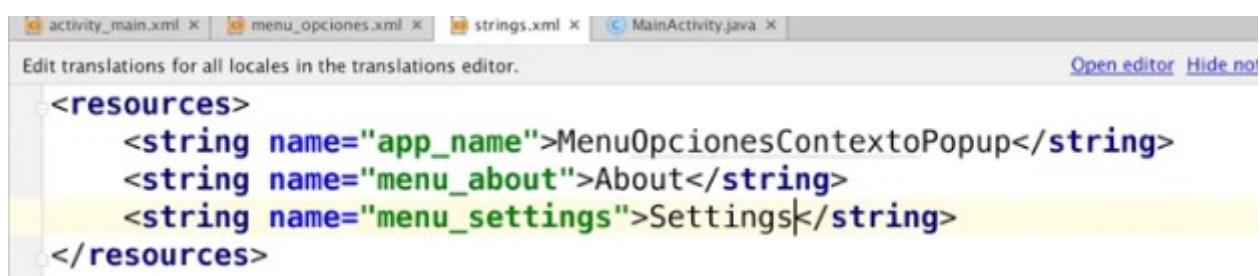
Mientras tanto, voy a declarar mi siguiente menú, voy a colocar Android:id y vamos a colocar el siguiente que @ + id/mSettings

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/mAbout"></item>
    <item android:id="@+id/mSettings">
        </item>
</menu>
```

Lo siguiente que tengo que colocarle a este menú, voy a empezar a dar un par de saltos de línea, para acomodar mejor mis etiquetas y mis propiedades.

Bien. Lo siguiente es el texto que va a decir mi menú.

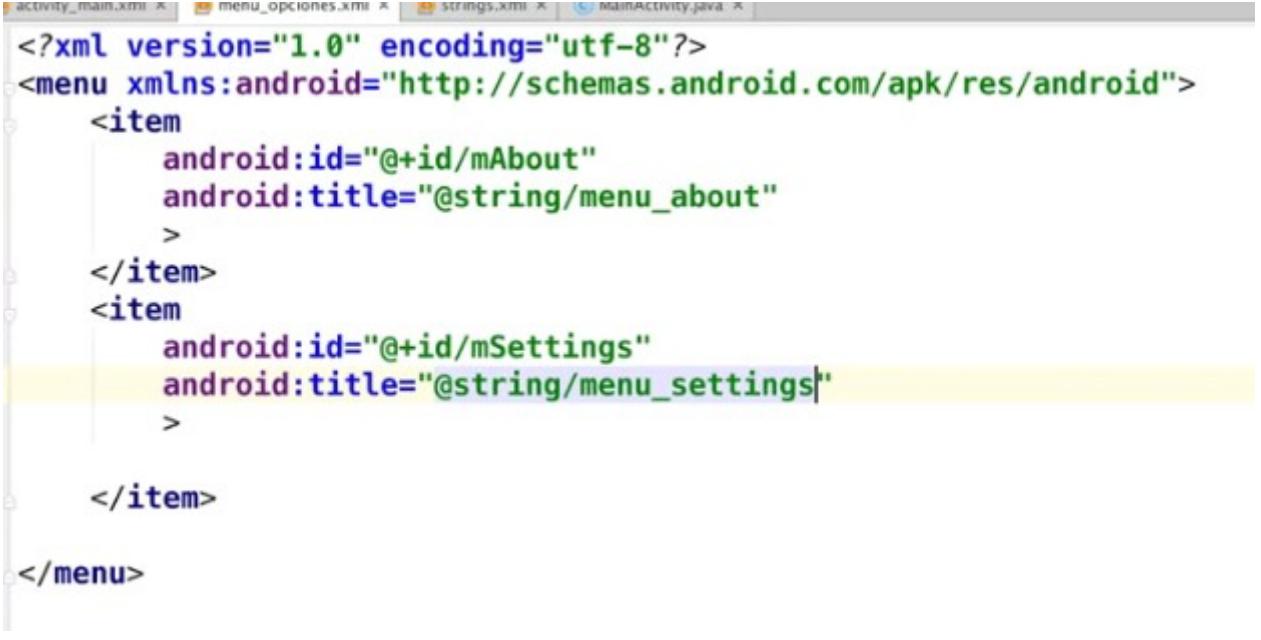
Entonces, voy a colocar la propiedad Android:title y iré a continuación a mi archivo Strings, a dar de alta precisamente los textos que van a responder a este menú.



```
<resources>
    <string name="app_name">MenuOpcionesContextoPopup</string>
    <string name="menu_about">About</string>
    <string name="menu_settings">Settings</string>
</resources>
```

Entonces, primero dijimos que debe ser menu_about y vamos a poner About. Posteriormente, tenemos nuestro menú de Settings y vamos a colocar Settings.

Entonces, regreso a mi archivo xml de menú de opciones y, con haciendo referencia a mi archivo Strings, voy a colocar el menú About y lo mismo para el menú Settings, Android title, menú Settings.



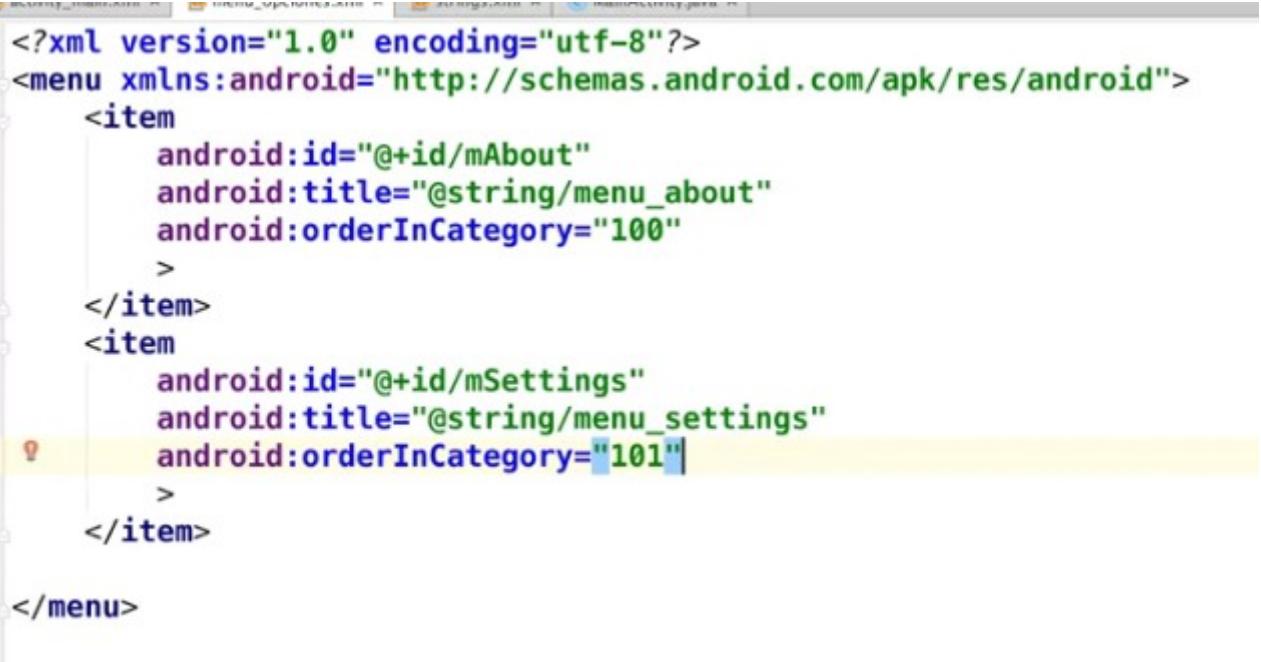
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/mAbout"
        android:title="@string/menu_about"
    >
    </item>
    <item
        android:id="@+id/mSettings"
        android:title="@string/menu_settings"
    >
    </item>
</menu>
```

Bien. Como observas, estos son atributos obligatorios. Por lo menos, el título es un atributo obligatorio que debe estar ubicado en la etiqueta item.

Entonces, voy a agregar un último parámetro, Android: orderInCategory y vamos a ponerle que el orden en la categoría, este atributo es básicamente para poder estar ordenando mis menús, tener un ordenamiento, cuál va a aparecer antes, cuál va a aparecer después.

Entonces, voy a ponerle que tenga 100.

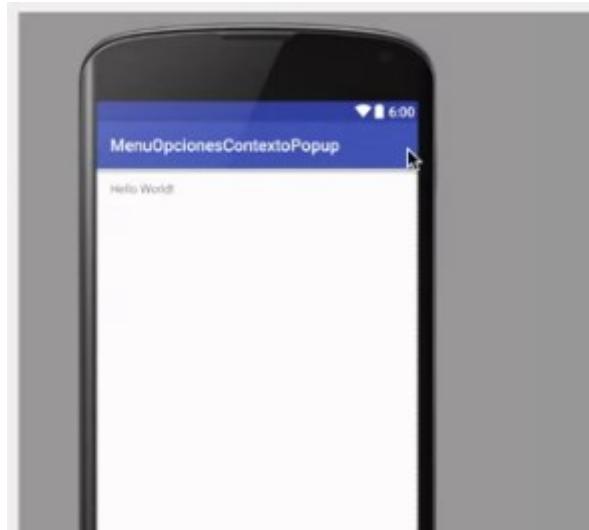
Y el otro, ese realmente es un número que tú puedes definir, que es libre, que puedes tener, básicamente, puedes basarte en lo que tú quieras para colocar este número. Entonces no hay ningún problema.



```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/mAbout"
        android:title="@string/menu_about"
        android:orderInCategory="100"
    >
    </item>
    <item
        android:id="@+id/mSettings"
        android:title="@string/menu_settings"
        android:orderInCategory="101"
    >
    </item>
</menu>
```

Ya tenemos aquí nuestros menús de opciones. Entonces, ahora debemos ir a implementarlos en nuestro Main Activity.

Tenemos nuestro Layout que hasta ahora está así, se ve de esta forma.



Y la idea es que nuestro menú de opciones va a aparecer, de pronto, tres puntitos en esta zona que está aquí y esto va a responder a nuestro menú de opciones.

Colocaremos en nuestro Main Activity.

Vamos a sobrescribir el método onCreateOptionsMenu, onCreateOptionsMenu.

```
package com.anncode.menuopcionescontextopopup;

import ...

public class MainActivity extends AppCompatActivity {

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

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

Este método, como su nombre lo dice, nos va a crear un menú de opciones.

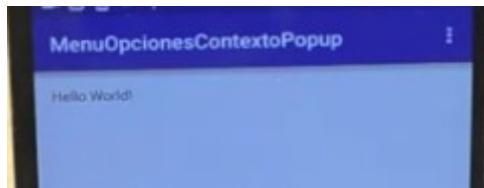
¿Y qué es lo que hará este método? Básicamente, tomar a nuestro layout que acabamos de crear nuestro recurso menú y lo va a inflar en la vista, lo va a mostrar en la vista.

Entonces, para esto, en vez de retornar esto que me está diciendo aquí, OnCreateOptionsMenu, voy a decirle getMenuInflater.inflate y lo que recibe aquí es el recurso que acabamos de definir R.Menu.menu de opciones.

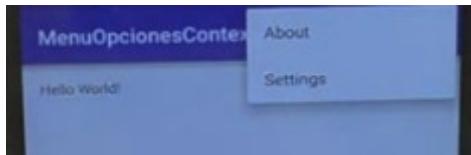
Aquí le vamos a colocar el objeto menú que estamos recibiendo por aquí, menú. Este que está aquí es lo que pasamos aquí. Y finalmente devolvemos True.

Si nosotros creamos, si nosotros corremos esto, ya podremos estar viendo un menú de opciones, vamos a correr nuestro proyecto y vamos a ponerlo por aquí.

Entonces, como observamos, ya podemos ver en esta zona, ya podemos ver nuestros tres puntitos y allí ya vemos nuestro menú About y Settings.



Bien. Aparece primeramente About y después Settings.



Vamos a colocar aquí un poco de movimiento, un poco de acción a estos dos opciones y vamos a controlar las opciones de este menú sobrescribiendo el método onOptionsItemSelected,

onOptionsItemSelected. Con este método, aquí es donde nosotros podemos controlar que si selecciono About o si selecciono Settings, qué es lo que va a suceder.

¿Entonces, cómo voy a controlar esto? Bien. Voy a controlarlo a partir del Id que corresponde a cada elemento.

Recuerdas que para About seleccionamos, definimos más bien, el Id mAbout.

Y para Settings definimos el mId, mSettings.

Entonces, A partir de estos Ids que definimos es como vamos a poder identificar cuál es el menú que alguien ha seleccionado. Entonces lo haremos a través de un switch.

The screenshot shows the Java code for the `onOptionsItemSelected` method in `MainActivity.java`. The code uses a `switch` statement to handle menu items based on their ID. It includes two `case` statements: one for the `mAbout` item which starts an `ActivityAbout`, and another for the `mSettings` item which starts an `ActivitySettings`. The code also includes a `return super.onOptionsItemSelected(item);` statement at the end.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.mAbout:
            Intent intent = new Intent(this, ActivityAbout.class);
            startActivity(intent);
            break;
        case R.id.mSettings:
            Intent i = new Intent(this, ActivitySettings.class);
            startActivity(i);
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

Tenemos que estar comparando cuál es el menú que seleccionaron.

Entonces tengo aquí mi switch, lo que va a recibir es si observas, tengo aquí el parámetro `MenuItem`. Lo que devuelve o más bien lo que recibe este método está recibiendo el ítem del menú seleccionado.

Entonces quiere decir que éste, quiere decir que cuando yo doy clic en `about`, ese elemento se pasa como parámetro en este método.

Entonces yo ya puedo preguntarle al ítem, puedo decir ítem, `getItemId`. Yo puedo preguntarle a ese ítem por su Id, y lo que me va a devolver, puedo poner `case R` punto `Id`. Y con otro `case` puedo decir si `R` punto `Id` es `mSettings`, el menú `settings`, entonces pues haz algo.

Entonces aquí dentro es donde bueno, si yo voy a ir a la ventana de `About`, pues yo puedo aquí levantar un intent, vamos a crear una segunda actividad, vamos a crear clic derecho `new empty activity`, y vamos a ponerle `Activity About`, y también vamos a crear nuestro `Activity Settings`.

Entonces simplemente para fines demostrativos vamos a añadir un par de `text views`, que simplemente digan `Activity About` y `Activity Settings`.

```
package com.anncode.menuopcionescontextopopup;

import ...

public class ActivityAbout extends AppCompatActivity {

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

```
package com.anncode.menuopcionescontextopopup;

import ...

public class ActivitySettings extends AppCompatActivity {

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

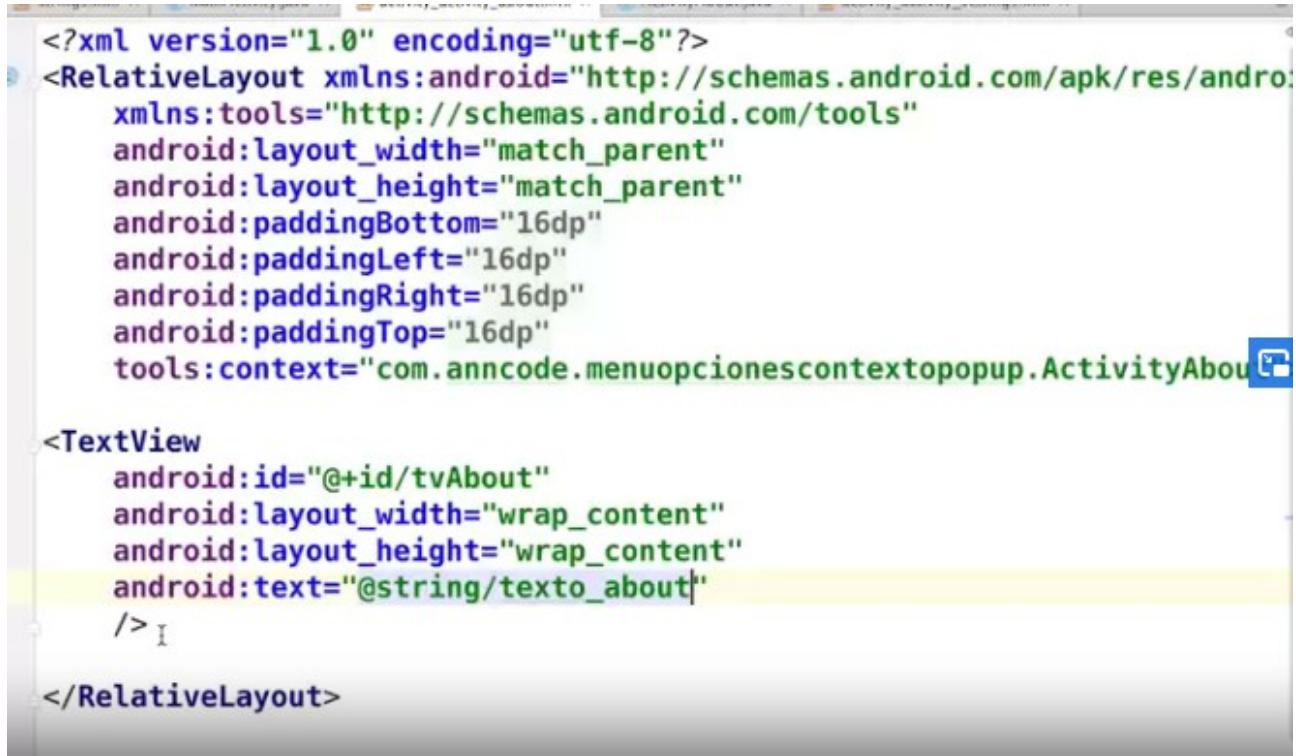
Entonces esos textos los voy a dar de alta en mi archivo, vamos a poner texto en archivo strings, vamos a poner texto about y vamos a poner About.

Podéis reciclar las variables anteriores de menú settings, pero la verdad es que como tienen diferente identificador, no me conviene, no me conviene para mi proyecto.

```
<resources>
    <string name="app_name">MenuOpcionesContextoPopup</string>
    <string name="menu_about">About</string>
    <string name="menu_settings">Settings</string>
    <string name="texto_about">About</string>
    <string name="texto_settings">Settings</string>
</resources>
```

Entonces simplemente voy a colocar aquí un TextView, vamos a ponerle wrap content, wrap content, ponemos su android id TextViewAbout. Y le ponemos su texto.

Y con eso se arma el xml para about y para settings



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.anncode.menuopcionescontextopopup.ActivityAbout">

    <TextView
        android:id="@+id/tvAbout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/texto_about"
    />

</RelativeLayout>
```

Entonces al darle About me está llevando a la pantalla de About.

Regreso, al darle Settings me está llevando a la pantalla de Settings.

De esa forma, es como yo puedo estar controlando los menús de opciones en mi proyecto.

Menú de contexto

Un menú de contexto es muy fácil de identificar. Es aquel menú que sucede cuando dejas presionado un par de segundos sobre la pantalla.

Ejemplo: Menú de contexto

Lo que voy a hacer es voy ir a mi activity_main.xml y recuerda que un menú de contexto es aquél menú que aparece cuando tú quedas bastante tiempo presionando un elemento, un largo tiempo presionando un elemento, y entonces sale el menú de contexto.

Lo que vamos a hacer es que este Hello world, mientras, si yo me quedo presionándolo un momento, de pronto, aparezca nuestro menú de contexto.

Entonces, vamos, primeramente, a declarar las opciones del menú de contexto.

Entonces ya tengo aquí mi carpeta menú, menú de opciones, recuerda que es para un menu_opciones, que aparece en la parte superior, ahora voy a darle clic derecho, New, Menu resource file y le voy a decir menú de contexto.

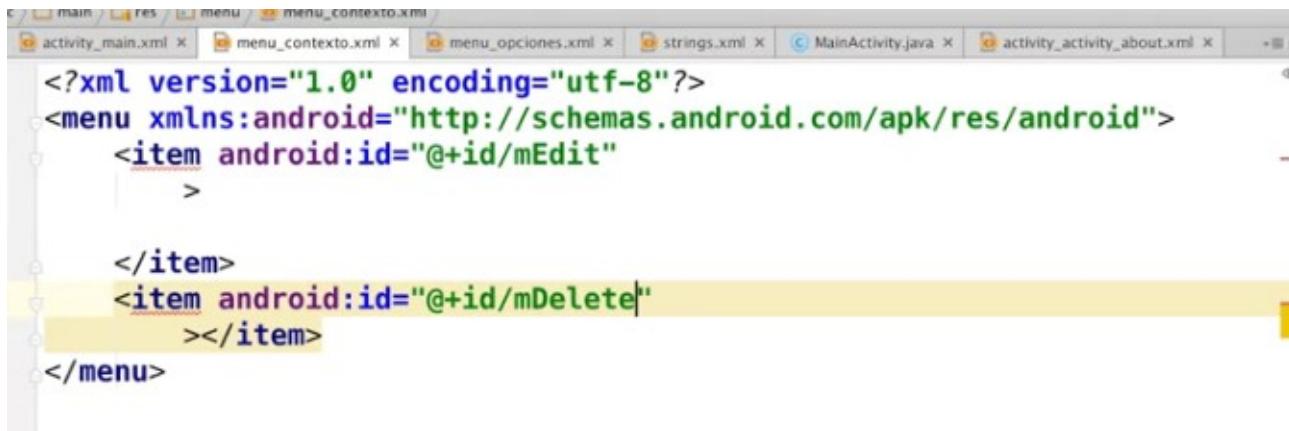
Ya tengo aquí mi menú de contexto, y entonces, únicamente voy a declarar un par de ítems.

Si quiero, vamos, nada más vamos a cambiar ese texto de Hello world, le vamos a poner el nombre de una persona y a lo mejor vamos a decir que si alguien deja presionado ese texto, pues puedes, por un lado, eliminar el elemento, o por otro lado, podrías editar el elemento.

Entonces vamos a declarar nuestro ítem, vamos a declarar dos ítems.

Recuerda, primeramente colocamos, android:id y colocaremos mEditar. Y al otro, le colocaremos android mEliminar.

Podríamos manejarlos en inglés, delete, nos viene aquí, Edit y Delete.



```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/mEdit">
    </item>
    <item android:id="@+id/mDelete">
    </item>
</menu>
```

Ahora, vamos a nuestro archivo strings para colocar los textos que deben tener estos menús.

Voy a mi archivo strings, strings, colocaré menu_edit, vamos a poner Edit string menu_delete, Delete.



```
<resources>
    <string name="app_name">MenuOpcionesContextoPopup</string>
    <string name="menu_about">About</string>
    <string name="menu_settings">Settings</string>
    <string name="texto_about">About</string>
    <string name="texto_settings">Settings</string>
    <string name="menu_edit">Edit</string>
    <string name="menu_delete">Delete</string>
</resources>
```

Regreso a mi menú de contexto, y ahora, con la propiedad android:title@string menu_edit, y el otro android:title @string menu_delete.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/mEdit"
        android:title="@string/menu_edit"
    >
    </item>
    <item
        android:id="@+id/mDelete"
        android:title="@string/menu_delete"
    ></item>
</menu>
```

Ya tengo mis dos menús listos.

Ahora vamos, precisamente, a darles vida a estos menús.

Vamos a nuestro archivo MainActivity. Entonces tenemos aquí nuestro archivo MainActivity, tenemos nuestros antiguos métodos onCreateOptionsMenu y onOptionsItemSelected.

Deabajo de ese método voy a sobreescibir el método onCreateContextMenu.

Aquí me aparece. Voy a dejar ese super tal cual como está, y este método lo que hará precisamente será crearnos o mostrarnos un menú de contexto, crear el menú de contexto cuando dejes presionado un largo tiempo algún view, algún elemento, puede ser un botón, puede ser un TextView, como lo vamos a hacer ahora, puede ser una imagen, puede ser lo que tú quieras.

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

Entonces, primeramente voy a colocar mi Menu Inflater.

Voy a colocar aquí inflater = new, MenuInflater y le vamos a pasar el contexto this. Ahora, a través de un switch, como la forma anterior donde tenemos nuestro onCreateOptionsMenu, también aquí vamos a getMenuInflater y vamos a inflar un menú, pero en nuestro caso un menú de contexto.

Entonces vamos a ponerle getMenuInflater.inflate y entonces va a recibir el recurso.

Vamos a poner R.menu.menu de contexto, ese es el menú que necesitamos, y, por último, pasamos el objeto menu, que, si observas en comparación con el anterior, el anterior era un objeto de tipo menú. Un menú simple, un menú por default. Este objeto que vamos a pasar es un menú de tipo contexto.

Aquí lo que estará haciendo, será que nos está creando un menú de contexto.

Entonces, ahora vamos a nuestro activity_main.

Vamos a hacer que este TextView pues esté preparado para levantar el menú de contexto.

El método que acabamos de crear.

Entonces, únicamente, voy a cambiarle primeramente el texto, a decir TextView, y le voy a poner texto , voy a poner nombre, nombre de una persona.

Vamos a nuestra vista del texto y le quitamos @string nombre de una persona.

Y entonces ahora sí vamos a poner este elemento listo para que levante un menú de contexto cuando lo presionemos varias veces.

Vamos a nuestro MainActivity que es quien controla este layout, MainActivity.

En nuestro método onCreate, nuestro método onCreate, sencillo, el que crea nuestra actividad.

Ahí vamos a colocar la instrucción, vamos a poner nuestro TextView, vamos a poner nuestro TextView en órbita. Entonces voy a poner tvNombre, qué text tiene, no le hemos colocado un Id, entonces, vamos a ponerle el Id, porque lo vamos a necesitar, Id tvNombre.

Regreso a mi Main Activity y ahora sí, you puedo hacer mi casting TextView findViewById R.id .tvNombre, y con la instrucción registerForContextMenu le paso el View, en nuestro caso tvNombre, ese TextView.

```
public class MainActivity extends AppCompatActivity {

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

        TextView tvNombre = (TextView) findViewById(R.id.tvNombre);
        registerForContextMenu(tvNombre);
    }
}
```

Y entonces, con esta instrucción, yo ya he colocado ese menú disponible. Ya he colocado este View disponible para que levante un menú de contexto

Nos falta, también, switchear, pues cuál es el menú que han dado clic.

Y para eso lo haremos con otro, otro método que veremos más adelante.

Entonces, vamos a sobreescibir el método onCreateItem, perdón, es onContext, onContextItemSelected.

O sea qué item de contexto he seleccionado y como observas aquí también me trae el MenuItem.

Y, exactamente utilizando la misma lógica que hicimos para nuestro menú de opciones, a través de un switch, podemos estar verificando cuál es el elemento que le dieron, que le dieron clic.

Le preguntamos al item recibido, le preguntamos cuál es su Id.

```
@Override  
public boolean onContextItemSelected(MenuItem item) {  
  
    switch (item.getItemId()) {  
  
        case R.id.mEdit:  
            //Intent  
            break;  
  
        case R.id.mDelete:  
            //Intent  
            break;  
  
    }  
}
```

Bien, y de la misma manera que en el anterior, aquí es donde puedo colocar mi Intent, a dónde quiero que vaya, o, no sé, probablemente no solamente esté yendo a otra pantalla en particular, probablemente sea no sé, descargar un archivo, ir a un sitio web, consultar una página, no sé, etcétera, lo que esté marcado para tu menú.

Todo eso lo colocamos dentro de esto.

Menú Pop-up

Un menú pop-up es aquel menú que vemos desplegarse en forma de lista cuando presionamos algún view, algún view de acción, puede estar en la barra o puede estar dentro de nuestra interfaz gráfica.

Un menú pop-up siempre va a aparecer en una lista sorpresivamente.

Ejemplo: Menú Popup

En este caso no tendremos que esperar, o no tendremos que dejar presionado sobre nuestro elemento para que aparezca, sino que simplemente dándole un touch, o dándole un tap a ese elemento inmediatamente se va a desplegar el menú.

Entonces, para hacer esto, voy a crear por aquí, yo el tipo de view para estar manejando otros tipos de elementos sobre menús, voy a manejar una imagen

Una imagen que previamente pues yo ya tengo importada en mi archivo drawable, y es esta imagen, este sol que es tomado de internet, de la página de Google. Y, bueno. Entonces, vamos a hacer que esa imagen, pues precisamente cuando le demos tap, cuando tapemos esa imagen, pues, se levante un menú.

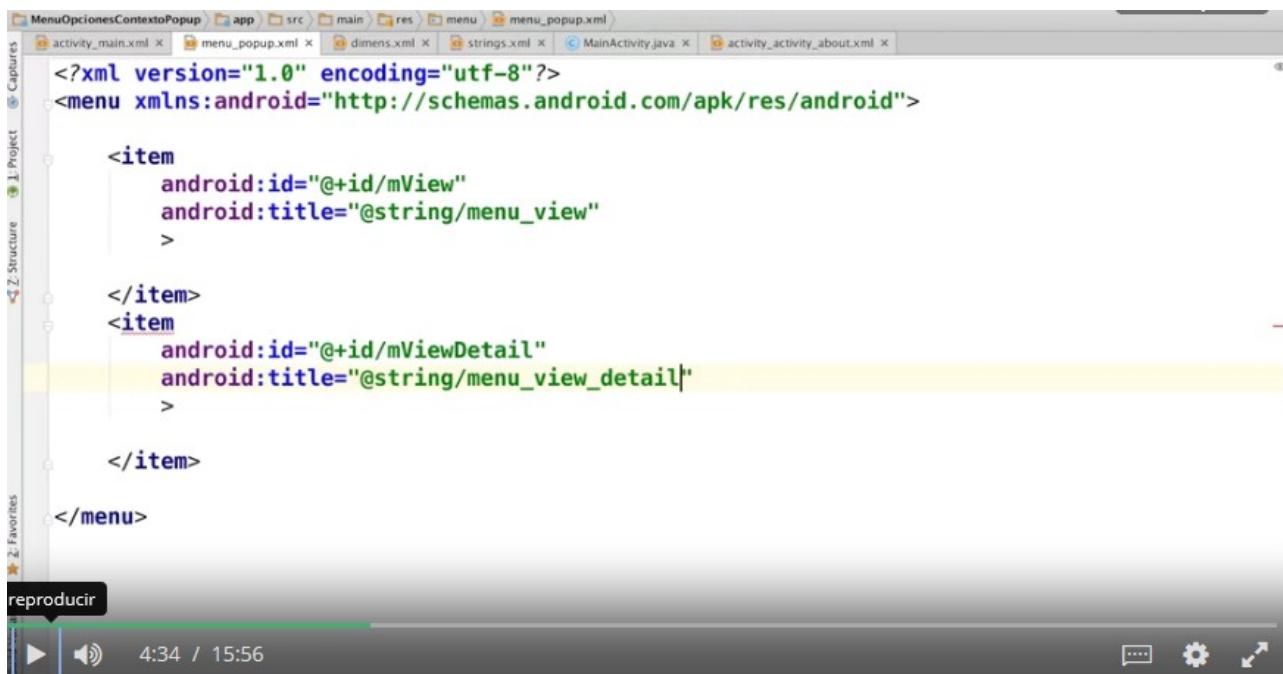
Podremos decir que el menú que queremos que se levante sea un menú, podría decir, visualizar la imagen o también, podríamos decir ver detalles de la imagen. Dos opciones en el menú.

Bien, aquí también puedo crear un nuevo menú. Ahora, para fines de nuestro ejemplo, yo estoy creando un menú para cada concepto que estamos viendo.

Pero realmente, si así lo quieres, todos tus menús podrían vivir en un solo archivo. Aunque yo te recomiendo que hagas esto, lo hagas de esta forma, para que puedas tener, sobre todo por, por mejor práctica, por una buena práctica en tu proyecto, que lo tengas mejor estructurado.

Vamos a poner el menu_popup, menu_popup. Le damos Finish, y ahora vamos a empezar a añadir nuestro menú popup.

Bien, voy a poner nuestro item de la misma forma como hemos estado manejando nuestros items, voy a darle aquí un enter, y también aquí, a cada elemento le vamos a poner su respectivo Id, y también su respectivo título android, recuerda primeramente el Id, le puedo poner menú, dijimos que uno será view y ahorita me voy por los títulos, android:id.



```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/mView"
        android:title="@string/menu_view"
    >

    </item>
    <item
        android:id="@+id/mViewDetail"
        android:title="@string/menu_view_detail"
    >

    </item>
</menu>
```

Podemos poner ViewDetail, ViewDetail, ver los detalles de esa imagen. Y ahora en mi archivo strings pues vamos a colocar los textos que va a decir este menú.

Recuerda que es una muy buena práctica tener los textos en nuestro archivo strings, puesto que cuando querremos manejar el soporte para múltiples idiomas pues será sumamente sencillo estar manejando, estar manejando todos, sí, todos nuestros archivos concentrados en un solo archivo.

The screenshot shows the Android Studio interface with the 'strings.xml' file open. The code is as follows:

```
<resources>
    <string name="app_name">MenuOpcionesContextoPopup</string>
    <string name="menu_about">About</string>
    <string name="menu_settings">Settings</string>
    <string name="texto_about">About</string>
    <string name="texto_settings">Settings</string>
    <string name="menu_edit">Edit</string>
    <string name="menu_delete">Delete</string>
    <string name="nombre_persona">Anahi Salgado</string>
    <string name="menu_view">View Image</string>
    <string name="menu_view_detail">View Image Detail</string>
</resources>
```

Entonces, vamos a meter la imagen. Vamos a colocar la imagen en nuestro layout.

Vamos a ponernos debajo de este TextView, y estoy en la vista del texto de mi activity_main. Lo que voy a insertar aquí no es un TextView sino es un ImageView.

Vamos a decirle wrap_content y wrap_content, vamos a ponerle android:id y será imgImagen, Imagen. Vamos a ponerle android:src, colocamos drawable y esa es nuestra imagen. Vamos a ver cómo está viéndose esto, ya aquí se alcanza a percibir.

Vamos a darle una posición a esto. Vamos a decir que esté centrado, vertical, y también que esté centrado horizontal. Veamos en el preview, aquí está, you se ve nuestra imagen.

The screenshot shows the Android Studio interface with the 'activity_main.xml' layout file open. The code is as follows:

```
<TextView
    android:id="@+id/tvNombre"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/nombre_persona"
    android:textSize="@dimen/texto_persona"
    />

<ImageView
    android:id="@+id/imgImagen"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/unnamed"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true"
    />
</RelativeLayout>
```

La idea es que, al pulsar la imagen, se despliegue un menú de ella.

Entonces para hacer esto, para hacer este menú popup, lo que tenemos que hacer es disparar un evento o disparar un método que reaccione cuando alguien le dé clic a esta imagen.

Entonces, al igual que en los botones, todos los views también tienen disponible esta propiedad android:onClick, Android:onClick.

Entonces lo que haremos será crear un método que dentro de ese método tendrá toda la lógica para levantar un popup, y ese método creado lo vamos a asociar a este atributo onClick.

Entonces, voy a ir al archivo de Java que controla este layout, MainActivity.

Entonces, voy a ponerle public void y le voy a poner menuPopUp.

Vamos a ponerle levantar MenuPopUp. Y entonces este va a recibir un view así, y aquí adentro vamos a tener toda la lógica para nuestro popup.

Para empezar, nuestro popup proviene de una clase, una clase PopUpMenu, y entonces debemos crear por aquí nuestro objeto. Vamos a ponerle igual a new PopUpMenu.

Vamos a poner el contexto. Nos pide el contexto de la actividad en donde será mostrado y posteriormente el view.

Vamos a colocar nuestro view.

Aquí vamos a revisar ahora si con este view es suficiente para levantar el menú, si no tendremos que declarar nuestro elemento ImageView y pasarlo como parámetro.

Entonces, voy a colocar aquí popupMenu .getMenuInflater.inflate, y como seguramente you debes saber pues lo que estoy haciendo es inflar el menú, a través del view que estamos seleccionando, a través del view que estamos levantando.

Voy a poner R.menu.menu_popup, le diré con mi objeto popup menu.getMenu

```
public void levantarMenuPopUp(View v){  
    PopupMenu popupMenu = new PopupMenu(this, v);  
    popupMenu.getMenuInflater().inflate(R.menu.menu_popup, popupMenu.getMenu());  
}
```

Y entonces pues este menú ya con estas líneas es suficiente, ya debería mostrarse nuestro menú. Y ahora nos faltarán, posteriormente, aquí nos falta este método, llevarlo al ImageView. Eso es muy importante. Debemos llevar el método a nuestro ImageView, y aquí nuestro proyecto ya se estaba preparando para correr.

```
<ImageView  
    android:id="@+id/imgImagen"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/unnamed"  
    android:layout_centerVertical="true"  
    android:layout_centerHorizontal="true"  
    android:onClick="levantarMenuPopUp"  
/>
```

Y antes de cualquier cosa, pues vamos a ver, you está corriendo, vamos a ver qué pasa si tecleamos nuestro menú.

Ahora no está respondiendo. Entonces, vamos a colocar nuestro ImageView por aquí. ImageView imagen. Y vamos a ponerle aquí su casteo ImageView, vamos a poner findViewById R.id.ImageView Imagen. Podemos pasar aquí el objeto imagen.

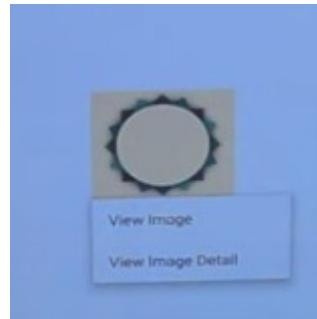
```
public void levantarMenuPopUp(View v){  
    ImageView imagen = (ImageView) findViewById(R.id.imgImagen);  
    PopupMenu popupMenu = new PopupMenu(this, imagen);  
    popupMenu.getMenuInflater().inflate(R.menu.menu_popup, popupMenu.getMenu());  
  
}
```

Aquí algo que nos está pasando es que debemos poner por supuesto mostrar nuestro menú.

Sí no lo mostramos, no se mostrará, si no utilizamos el método show, no se mostrará nuestro menú.

```
public void levantarMenuPopUp(View v){  
    ImageView imagen = (ImageView) findViewById(R.id.imgImagen);  
    PopupMenu popupMenu = new PopupMenu(this, imagen);  
    popupMenu.getMenuInflater().inflate(R.menu.menu_popup, popupMenu.getMenu());  
  
    popupMenu.show();  
  
}
```

Ahora sí, ahí está nuestro menú, View Image, View Image Detail.



Si observas, simplemente al darle un clic se muestra el menú.

Ahora solamente nos falta que, pues sobre este View, View Image o View Image Detail, sobre este View, pues podamos cachar cuando alguien le de clic a cualquiera de estos elementos.

Es muy sencillo, el objeto `popupMenu` tiene un método `setOnMenuItemClickListener`, tiene un método `listener` que siempre está alerta y está escuchando cuando tú le des touch o cuando le des tap a alguno de estos elementos.

Entonces lo que haré es utilizar este y vamos a crear una interfaz new `OnMenuItemClickListener`, y como observas tengo aquí `MenuItemClick` y, también estoy recibiendo un `MenuItem` y por supuesto puedo estar utilizando igual un `switch` para estarle preguntando al item cuál es el Id que estás recibiendo.

Puedo poner mi primer caso `R.id.view`,

Break. Y el segundo `R.Id.mViewDetail` y colocar break también.

Y podemos devolver true, no olvides devolver true aquí.

Entonces para poder ver un poco más lo que está sucediendo, puedo colocar de la misma forma un `toast`, coloco `this`, nuestro texto `getResources.getString`, colocar `R.string.menu_view`, `Toast.length_long` y no olvides nuestro `.show` que es muy importante.

Okey, aquí me está diciendo que no estoy en un contexto de actividad, entonces le pondré `getBaseContext`, listo.

Y es verdad, no estoy en un contexto de actividad, por esto que me encuentro dentro de una interfaz que está sobreescritiendo el método `OnItemClick`. Entonces simplemente aquí vamos a poner `viewDetail`.

```
public void levantarMenuPopUp(View v){
    ImageView imagen = (ImageView) findViewById(R.id.imgImagen);
    PopupMenu popupMenu = new PopupMenu(this, imagen);
    popupMenu.getMenuInflater().inflate(R.menu.menu_popup, popupMenu.getMenu());

    popupMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem item) {

            switch (item.getItemId()) {
                case R.id.mView:
                    Toast.makeText(this, getResources().getString(R.string.menu_view), Toast.LENGTH_LONG).show();
                    break;

                case R.id.mViewDetail:
                    break;
            }
            return true;
        }
    });
}
```

Vamos a correrlo, y esta es nuestro último menú.

Aquí está nuestro menú y ahí está respondiendo View Image, puedo seleccionar el otro y ahí está View Image Detail.

Integrando Action Views

Una vez que ya tienes toda la teoría de cómo funcionan los menús, verás que es muy sencillo integrar ActionViews en el AppBar de tu proyecto, solo tenemos que hacer un par de ajustes.

1. Añade el item en uno de tus archivos de recursos, yo lo he agregado en el archivo de menú que hemos trabajado menu_opciones.xml de la siguiente forma:

```
<item
    android:id="@+id/mRefresh"
    android:title="Refresh"
    android:orderInCategory="102"
    app:showAsAction="always"
    android:icon="@drawable/ic_refresh" />
```

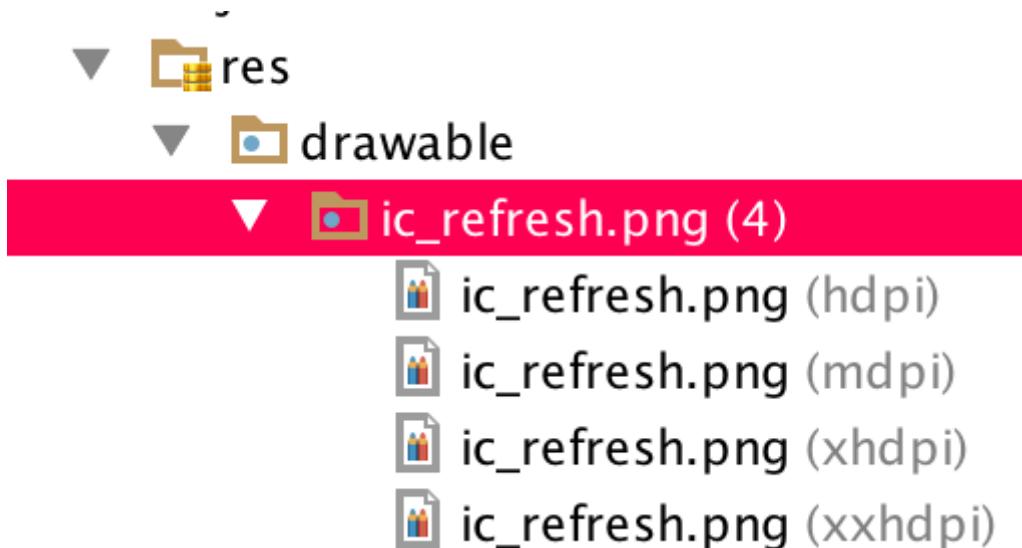
La propiedad app:showAsAction="always" es la que define que tu menú aparezca como un Action View en tu AppBar, por lo demás son las propiedades que hemos venido manejando en todos los menús.

Es muy importante que importes el name space de app en la cabecera del archivo, como se muestra a continuación

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
```

2. Hemos añadido un ícono de Refresh el cual está reproducido en sus diferentes densidades y en sus respectivos directorios.

Hemos añadido un ícono de **Refresh** el cual está reproducido en sus diferentes densidades y en sus respectivos directorios.



3. Recuerda que para que este botón ejecute una acción especial, debes sobreescibir el método `onOptionsItemSelected` y a través de un switch o un if obtener el id del item, como se muestra a continuación:

Recuerda que para que este botón ejecute una acción especial, debes sobreescibir el método `onOptionsItemSelected` y a través de un switch o un if obtener el id del item, como se muestra a continuación:

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
  
    switch (item.getItemId()) {  
  
        case R.id.mRefresh:  
  
            break;  
    }  
  
    return super.onOptionsItemSelected(item);  
}
```

Fragments

¿Por qué usar Fragments?

Los fragments se incluyeron en Android a partir de la versión 3.0, a partir de Honeycomb.

Como casi todo lo que hemos visto últimamente, la versión de Android 3.0 trajo muchos cambios a toda la estructura, a toda la interfaz de cómo vemos ahora las aplicaciones.

A partir de la versión 3.0 aparecieron las tabletas y con ello trajo una mayor cantidad de espacio para manejar nuestras interfaces gráficas.

Por eso dio origen a los fragments, los fragments ayudarán a que manejes mejor toda la proporción de interfaz dentro de tu aplicación móvil.

De tal forma que no estemos desperdiciando espacio, en toda la pantalla y que la aprovechamos mejor.

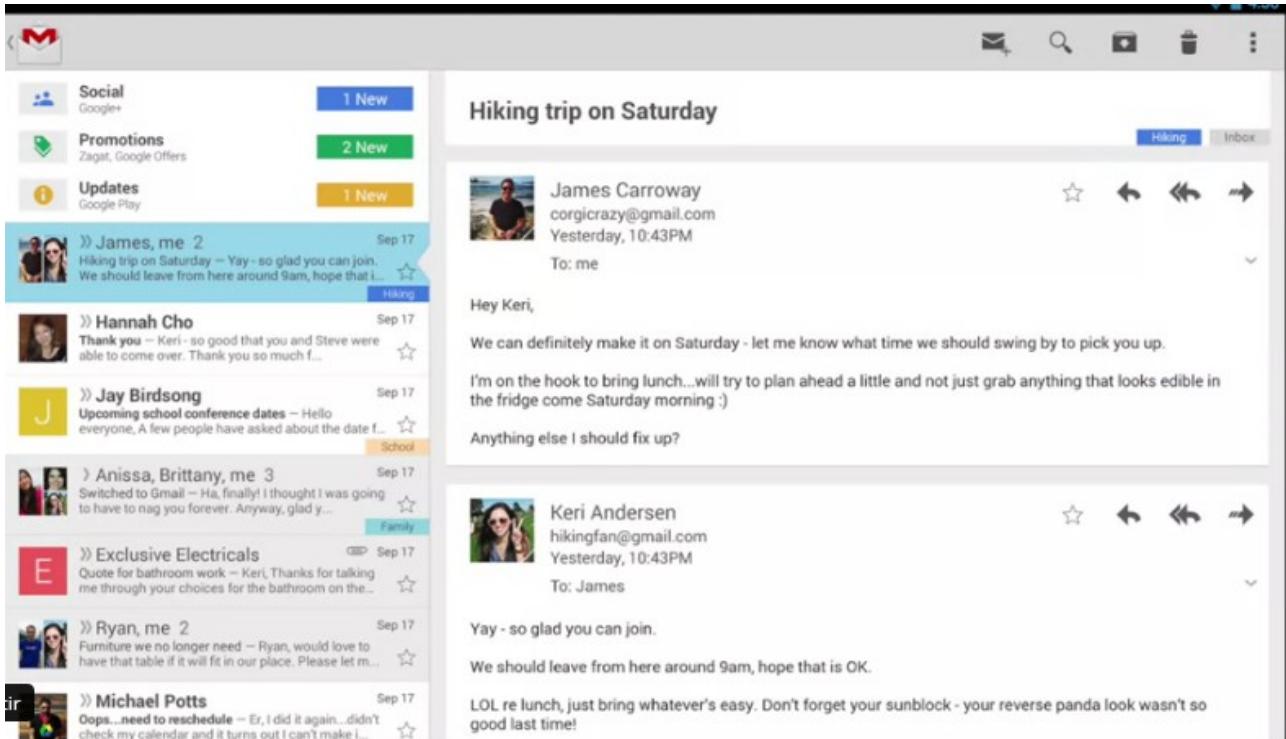
Como puedes observar por acá en esta imagen, tenemos una lista de correos. Una lista de correos que si estamos viendo con nuestro teléfono pues simplemente se mostrará la lista de correos.



¿Pero qué sucederá si estamos viendo ahora la lista de correos con nuestra tableta? Bueno, el manejar una lista de correos en una tableta de esta forma sería desperdiciar mucho espacio.

Entonces en Android se introdujeron unos fragments precisamente para esto. Para poder proporcionar mejor la interfaz.

Como vemos ahora en una tableta, si queremos leer nuestros correos electrónicos, tenemos por un lado la lista de correos electrónicos y por otro lado tendremos el detalle de los correos electrónicos.



Es decir, puedes aprovechar mejor la distribución de la pantalla y entonces en una sola pantalla puedes tener la lista de contactos y al mismo tiempo poder estar leyendo tus correos electrónicos.

Esto dio origen a los fragments.

Gracias a los fragments podemos tener interfaces mejor distribuidas sobre pantallas mucho más grandes.

Un fragment, al igual que los activity, necesitará esencialmente de dos componentes.

Primeramente un archivo xml, donde ahí estarás definiendo toda tu interfaz gráfica, todos tus views, todos tus componentes.

Y también una clase de Java, que en este caso esta clase estará heredando no de la clase activity, sino de la clase fragment.

Por esa razón podemos decir que un fragment no es una activity, dado que cada clase hereda de diferentes clases padre.

Una activity por una parte, hereda de la clase activity y un fragment por otra parte, heredará de su clase padre, fragment.

En los fragments es muy importante que nunca pierdas de vista estas dos clases. [La clase fragment manager y la clase fragment transaction](#).

Estas dos clases te van a ayudar precisamente a integrar tu fragment en una actividad.

La clase **fragment manager**, por un lado, nos ayudará a manejar todas las secciones, toda la manipulación que podemos tener sobre nuestro fragment.

Y la clase **fragment transaction** nos ayudará como un auxiliar para poder manipular todas las operaciones que la clase fragment va a hacer.

Se dice que cada operación que realice la clase fragment se considerará una transacción. Por eso podemos utilizar esta clase fragment transaction, que nos ayudará a manipular todas las transacciones que tengamos en nuestro fragment.

Finalmente, una vez que ya tenemos nuestras dos clases, fragment manager y fragment transaction. Finalmente utilizaremos el método oncreate de la clase activity, para precisamente ahí, incrustar nuestro fragment, y darle vida a nuestro fragment.

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        getSupportFragmentManager().beginTransaction()  
            .add(R.id.mifragment, new MiFragment())  
            .commit();  
    }  
}
```

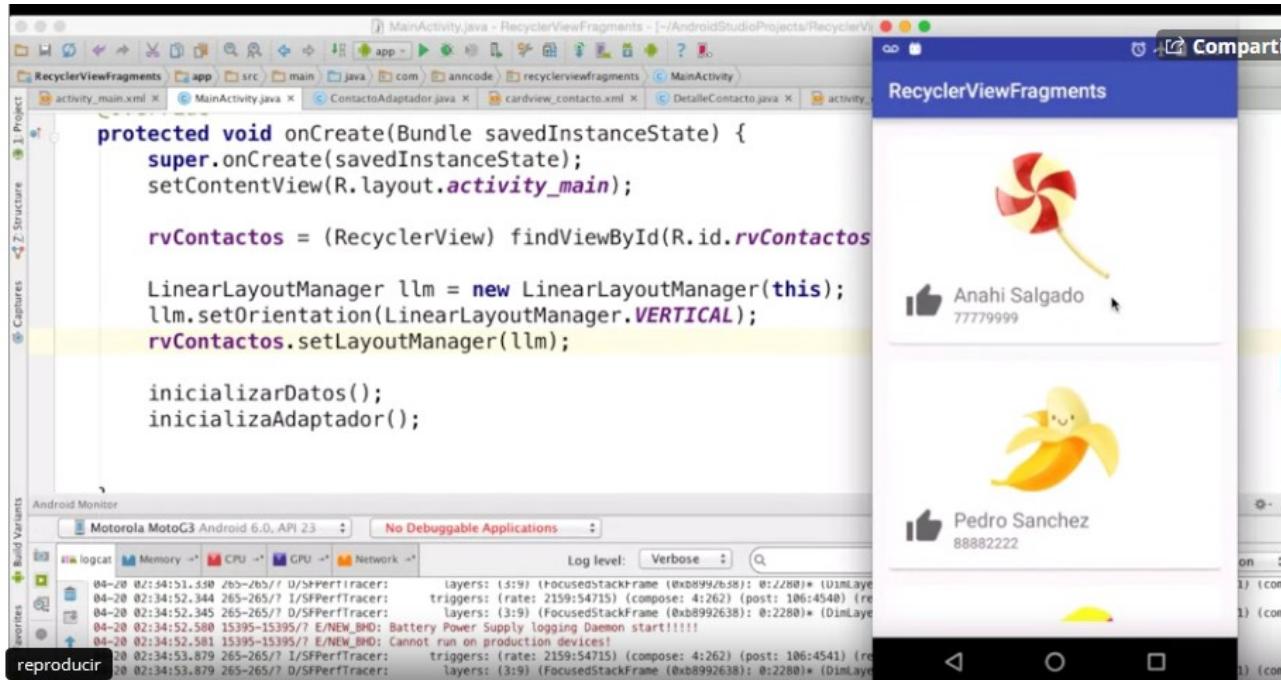
Podemos decir que el ciclo de vida de un fragment va a depender completamente del ciclo de vida que tenga un activity. Si un activity se detiene, el fragment también se detendrá.

Si una actividad se destruye, el fragment también se destruirá. ¿Por qué? Porque un fragment la mayoría de las veces siempre estará incrustado en una actividad.

Creación de Fragments

Me encuentro en este momento en la última aplicación que hemos hecho.

Donde tenemos un RecyclerView. Este RecyclerView al darle clic en la imagen lo que hace es que nos pasa a una actividad donde mostramos los detalles de ese contacto.



Es una aplicación de contactos, entonces puedo ir con Juan López, me muestra su número telefónico y su correo electrónico.

Y si le doy clic o si le doy like a una de estas fotos, a uno de estos contactos, me aparece un mensaje que dice, diste like a Pedro Sánchez, diste like a Anahí Salgado.

Y así puedo estar manejando el like de cada card o de cada tarjeta que está incrustada en el RecyclerView.

Entonces me encuentro aquí en el código, en nuestro proyecto.

Y lo que vamos a hacer ahora, es que vamos primeramente a manejar todo ésto en términos de fragments.

Entonces para este ejemplo, lo que vamos a ver es cómo podemos crear un fragment, cómo podemos hacer la ejecución o cómo podemos tener un nuevo fragment dentro de nuestro proyecto.

Este es el proyecto que tenemos, estoy en la vista de Android. Y entonces para crear un fragment.

Un fragment, como ya acabamos de ver, se va a componer de un archivo de Java y de un archivo XML.

Así que voy a comenzar creando mi archivo XML, voy a colocarle layout resource file.

Y le voy a poner fragment. Fragment RecyclerView. Este fragment lo que va a hacer es que va a contener un RecyclerView.

Lo vamos a dejar así como está y ahora vamos a manejar la clase que va a estar controlando este layout. Entonces voy a darle un clic derecho en Java class. Y voy a ponerle RecyclerView fragment.

Ahora, lo que tenemos que hacer a continuación es heredar de la clase fragment.

Y nosotros vamos a estar heredando de esta clase, de esta paquetería, android.support.v4.

Tenemos la anterior, pero nosotros como queremos estar manejando el soporte para versiones anteriores al API versiones anteriores a Android 3, entonces vamos a estar manejando esta librería.

Ahora, los métodos que tenemos que estar trabajando son, vamos aquí a sobrescribir el método onCreateView.

Y el método onCreateView va tener, vamos a comentar este return. Y lo que va a contener el método onCreateView va a ser precisamente la inflación, inflar nuestro layout con nuestro fragment.

```
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

/**
 * Created by anahisalgado on 20/04/16.
 */
public class RecyclerViewFragment extends Fragment {

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        //return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Lo que vamos a colocar aquí, vamos a poner un view y vamos a asignarle el layout inflado a ese view. Entonces vamos a decirle R.layout.fragment_recyclerview.

Le decimos que ésto va a ser un contenedor, se va a comportar como un contenedor y vamos a colocarle aquí un false.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    //return super.onCreateView(inflater, container, savedInstanceState);
    View v = inflater.inflate(R.layout.fragment_recyclerview, container, false);
}
```

Entonces esta línea que tenemos por aquí, esa línea equivaldría a lo que hemos venido trabajando con nuestro main activity. Equivaldría al setContentView.

En términos de una actividad, la forma en como nosotros asociamos un layout a una clase activity, lo hacemos a través de setear la vista con un subcontent view.

Pero en el fragment lo que usamos es el concepto de inflación. Y entonces esta línea que está aquí equivaldría a asignarle esta clase de Java a este layout. Y entonces ahora todo nuestro layout está convertido o está dentro de este pequeño objeto.

Entonces únicamente lo que vamos a estar haciendo es que vamos a devolver ese objeto y listo.

```

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    //return super.onCreateView(inflater, container, savedInstanceState);
    View v = inflater.inflate(R.layout.fragment_recyclerview, container, false);
    return v;
}

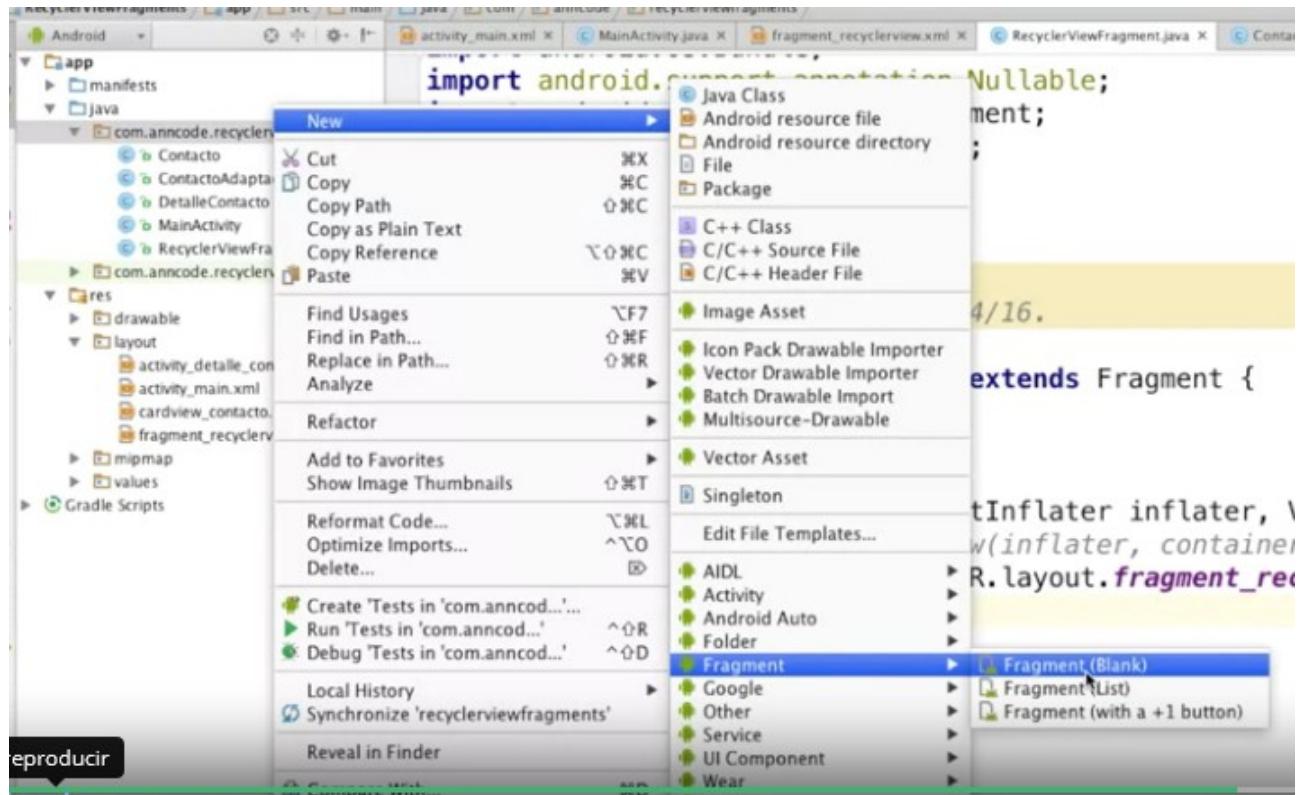
```

Con esto hasta el momento ya tengo listo mi fragment, ya mi fragment ya está asociado a ese RecyclerView.

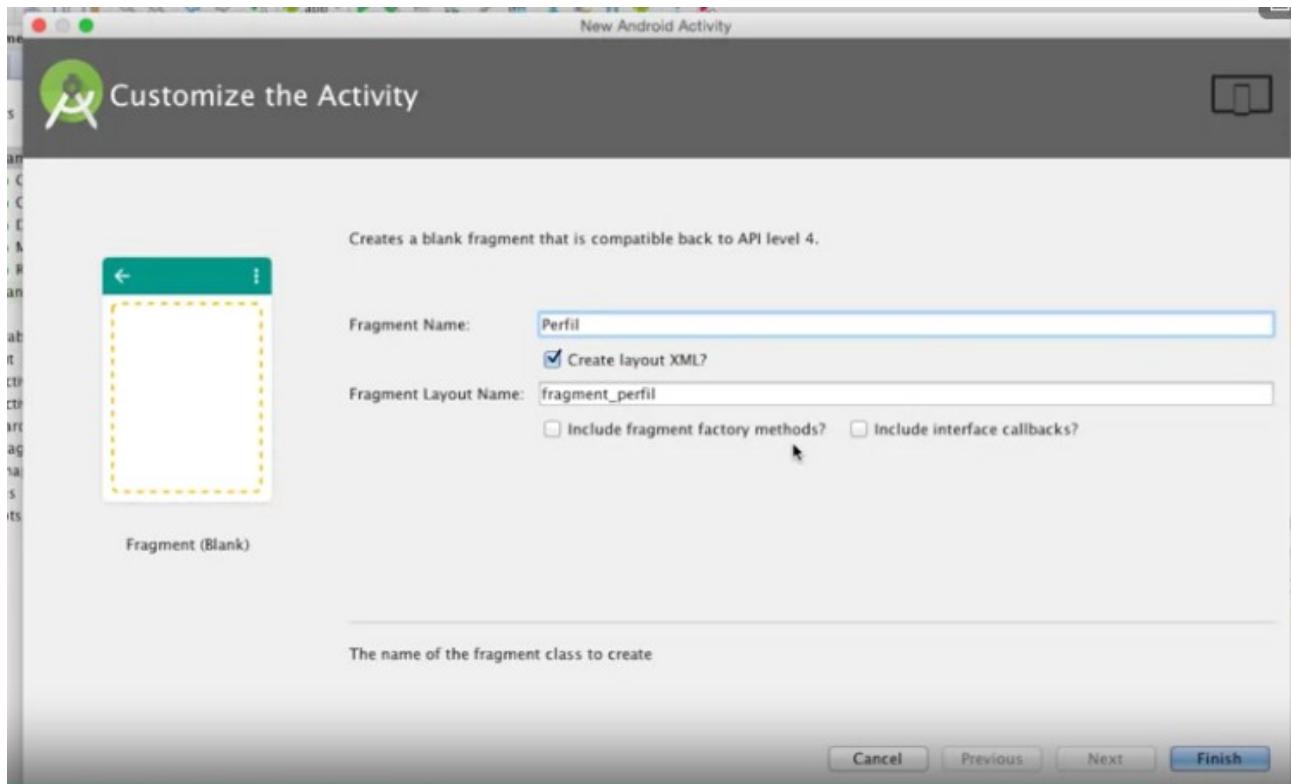
Y esa es una forma de estar creando fragments, digamos de una forma desglosada, de una forma manual.

Otra forma que también tenemos es que podemos crear fragments con ayuda del Wizard.

Podemos darle clic derecho y entonces podemos seleccionar aquí fragment.



Vamos a seleccionar un fragment en blanco y vamos a decirle que nos quite todos estos métodos callbacks para que no nos meta demasiado código.



Y lo que vamos a estar trabajando aquí en el fragment name. Aquí vamos a ponerle Perfil, porque es lo que vamos a estar trabajando en un segundo fragment. Entonces vamos a manejar PerfilFragment. Y en layout name vamos a manejar Fragment_perfil. Perfecto, vamos a dar finish.

Y en este momento pues el Android empieza a construir y empieza a crear nuestro proyecto.

Y lo que está haciendo en este momento es lo mismo que nosotros hicimos.

Simplemente crear una clase, hacer que esta clase esté heredando de fragment, y fragment está heredando de la versión cuatro. Estamos creando por aquí en el método onCreateView, estamos inflando nuestro layout fragment_perfil que es nuestro archivo, nuestro layout que tenemos por acá. Y lo está inflando.

Entonces aquí simplemente está devolviendo un return directo. En nuestro caso lo que hicimos fue almacenarlo en un objeto view y después devolver el objeto.

```
* A simple {@link Fragment} subclass.  
*/  
public class PerfilFragment extends Fragment {  
  
    public PerfilFragment() {  
        // Required empty public constructor  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
            Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.fragment_perfil, container, false);  
    }  
}
```

Pero aquí si observamos el fragment_perfil, de hecho ya nos está colocando incluso un text view. Y eso está dentro de un FrameLayout.

Éstas son dos maneras que tenemos para estar creando fragments en Android.

ViewPager y Fragment

Entonces una vez creados tus nuestros dos fragments, creamos un fragment que se llama RecyclerViewFragments.

Estos dos fragments los vamos a estar trabajando en un elemento que es un elemento nuevo que proviene de material design y este elemento se llama viewPager.

Vamos a estar manejando un viewPager, la idea de un viewPager es que podamos hacer un slide en esta forma y entonces podamos estar teniendo de primera vista nuestro recyclerView.

Y damos como un scroll de derecha a izquierda, entonces vamos a tener la vista de otro fragment o de otra actividad incrustada. O de otro fragment incrustado en la misma actividad.

Esa es la idea de los fragments, estar de alguna forma aprovechando la actividad que tenemos en este momento.

Y entonces poder estar haciendo vistas dinámicas de una forma muy vistosa, muy linda.

Entonces tenemos nuestro fragment éste es nuestro recyclerView. Hasta ahora este recyclerView está siendo mostrado en el main activity.

Ok, entonces lo que vamos a hacer es que vamos a transformar este main activity.

Vamos a transformarlo, vamos aadir un viewPager. Un nuevo elemento viewPager. Entonces para hacer esto, en nuestro main activity.

Aquí lo tenemos, este es nuestro layout activity main y tiene un tipo de layout, RelativeLayout.

Lo que yo voy a necesitar para poder estar manejando este viewPager, es que voy a necesitar otro tipo de layout que en este caso se llama CoordinatorLayout.

El CoordinatorLayout me va a ayudar a poder trabajar animaciones, transiciones y además agregar nuevos elementos que son muy vistosos como el viewPager.

Entonces puedo agregar un coordinator layout, pero si observas pues no aparece por ahí.

Y esto es porque para poder estar trabajando con este elemento, yo necesito traer una librería adicional. Esta librería es la librería de diseño. Entonces lo que haremos será traer a librería de diseño en nuestro archivo Gradle.

Entonces voy a colocar aquí, voy a hacer rápidamente un ajuste. Puedo traer la librería del enlace que hemos venido trabajando, pero en este momento voy a hacerlo de una forma más sencilla.

Simplemente voy a colocar este ajuste voy a verificar que los número de APIs correspondan y vamos por aquí a guardar nuestro proyecto.

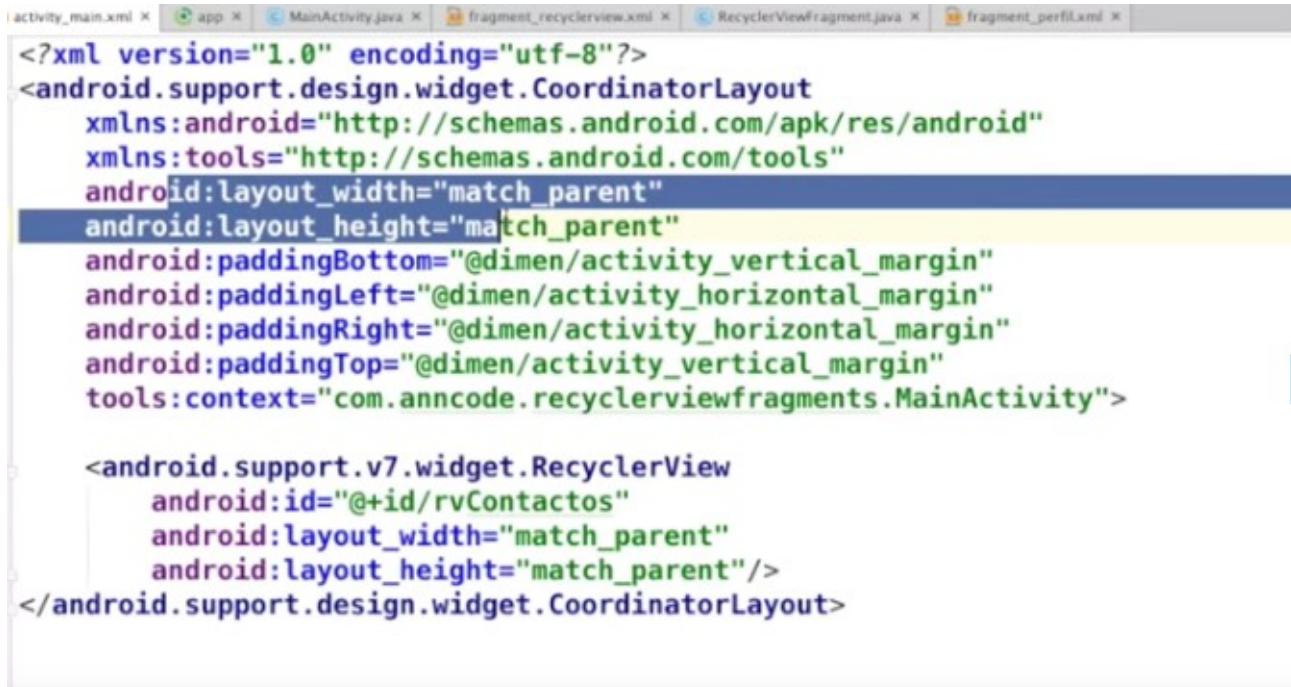
```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    testCompile 'junit:junit:4.12'  
    compile 'com.android.support:appcompat-v7:23.2.1'  
    compile 'com.android.support:cardview-v7:23.2.1'  
    compile 'com.android.support:recyclerview-v7:23.2.1'  
    compile 'com.android.support:support-v4:23.2.1'  
    compile 'com.android.support:design:23.2.1'  
}  
  
A newer version of com.android.support:design than 23.2.1 is available: 24.0.0-alpha1 [m]
```

Vamos hacer un rebuild on build y porque aquí me está mostrando este mensaje que probablemente hubo errores en Gradle. Pero si no te sale este mensaje pues puedes hacer la sincronización como venimos haciéndola siempre.

Entonces vamos a regresar a nuestro activity main, para precisamente estar manejando ahí nuestro layout coordinador.

Vamos a poner por aquí, vamos a escribir. Coordinator layout.

Y vamos a verificar que todo esté correcto.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.anncode.recyclerviewfragments.MainActivity">

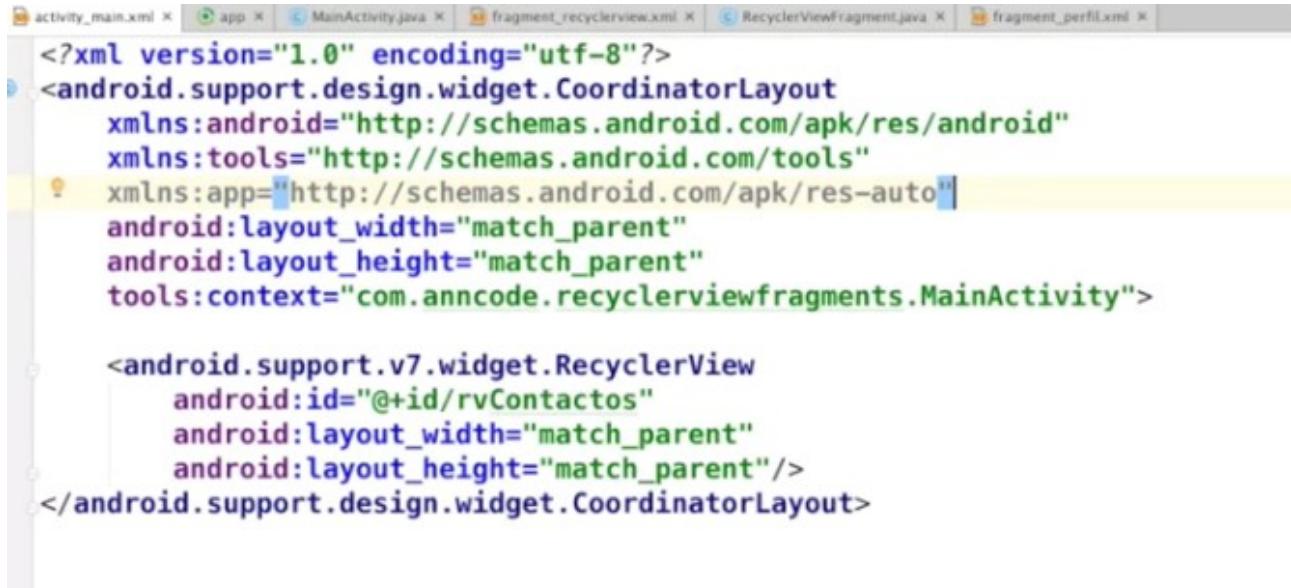
    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvContactos"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</android.support.design.widget.CoordinatorLayout>
```

Tenemos los name space, están bien. Tenemos los atributos obligatorios son width and height. Estamos colocando unos paddings, estos paddings los vamos a quitar.

Y vamos a colocar un name space adicional, que nos va a ayudar que es para la etiqueta app.

Entonces voy a copiar esto que está aquí, lo voy a pegar aquí, le voy a poner app, y listo.

Vamos a hacer que esto nos lo ponga así. Listo, simplemente escribiendo app y solito nos va traer esto.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.anncode.recyclerviewfragments.MainActivity">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvContactos"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</android.support.design.widget.CoordinatorLayout>
```

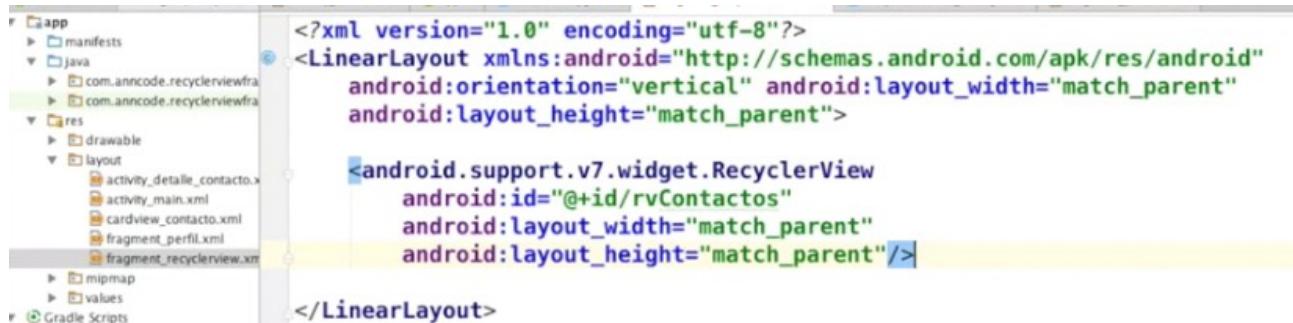
Entonces con esto ya quedaría listo nuestro coordinator layout.

Entonces, lo que a continuación vamos a hacer es este RecyclerView vamos a cortarlo, y lo que haremos será llevarlo a donde debe estar.

Ese RecyclerView tiene que estar en nuestro fragment RecyclerView, ok.

Entonces ahí debe estar nuestro RecyclerView. Lo vamos a cortar y lo vamos a poner en nuestro fragment RecyclerView. Vamos a ir a la vista de texto.

Entonces aquí lo vamos a poner.



The screenshot shows the Android Studio project structure on the left and the XML code editor on the right. The code is for a fragment named 'fragment_recyclerview.xml'. It contains a LinearLayout with a vertical orientation, a width of 'match_parent', and a height of 'match_parent'. Inside this layout is a RecyclerView with the following attributes: android:id="@+id/rvContactos", android:layout_width="match_parent", and android:layout_height="match_parent". The XML ends with a closing tag. The code is color-coded for syntax highlighting.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvContactos"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

Ok, con esto es suficiente en nuestro RecyclerView.

Entonces ya hemos sacado el RecyclerView. Lo hemos puesto fragment.

Recuerda que tanto este fragment RecyclerView como este fragment perfil, pues ambos deben estar viviendo en nuestro main activity, ambos. Por eso son fragments, son dos fragmentos, dos porciones de interfaz gráfica con clases que tienen funcionalidad propia, que están incrustados en un solo activity. Entonces de esta forma nos ayuda a poder estar aprovechando al máximo la interfaz de pantalla. La pantalla que tenemos en ese momento.

Entonces ya tenemos aquí nuestro coordinator layout, y entonces ahí mismo en el coordinator layout vamos a empezar a insertar nuestro viewPager.

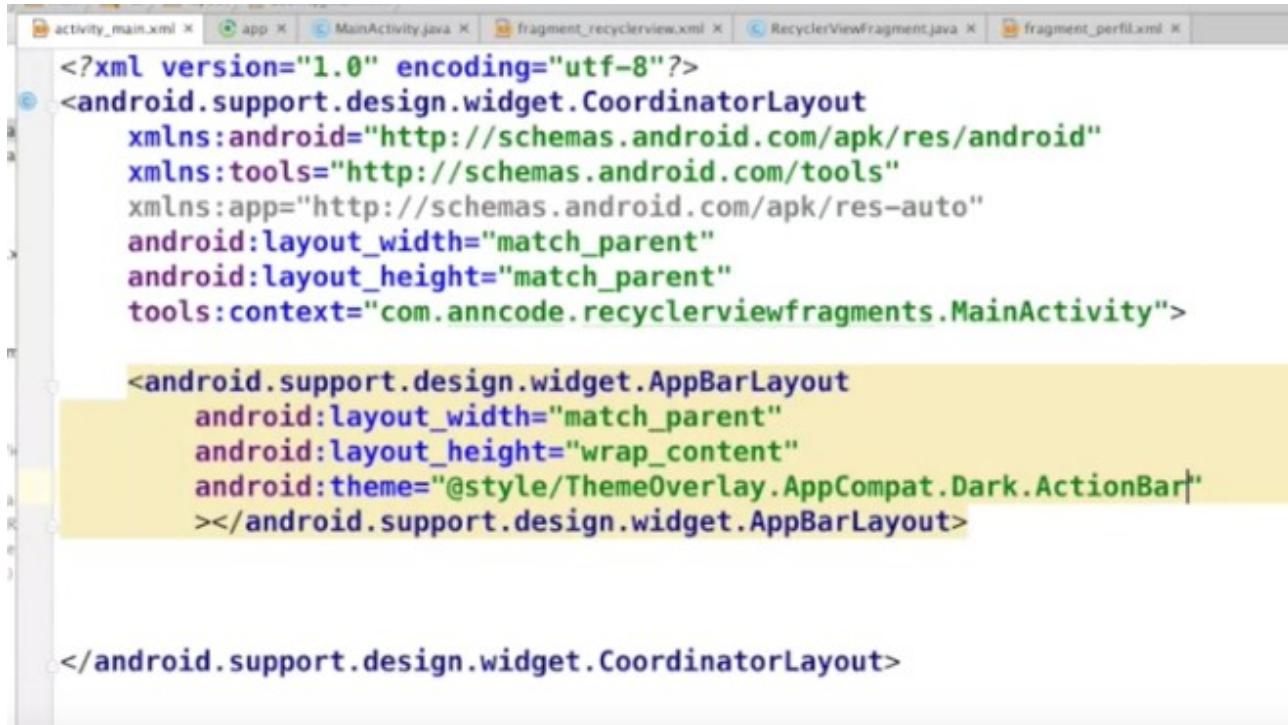
Primeramente vamos a colocar antes de nuestro viewPager, vamos a colocar nuestro app bar.

Entonces voy a poner este widget, support.design.widget.AppBarLayout.

Nuestro app bar vamos a decirle match_parent. Wrap content y vamos a colocarle un tema android:theme. Este trabajo probablemente pudiste haberlo ahorrado, simplemente creando un nuevo activity. Un nuevo activity que tuviera de esta forma.

Vamos a verlo por aquí, create new, activity, un activity en blanco, blank activity. Un blank activity you nos crea por default un coordinator layout y además nos crea por ahí un floating action button, etc.

Entonces esto es lo que necesito Compat.Dark.ActionBar. Ok, eso es lo que necesito para el tema de la barra. Esto estamos configurando la barra, nuestro AppBar, y dentro vamos a colocar toolbar. Y vamos a buscar así rápidamente toolbar. Vamos a ponerle un width match parent y un height de wrap content.

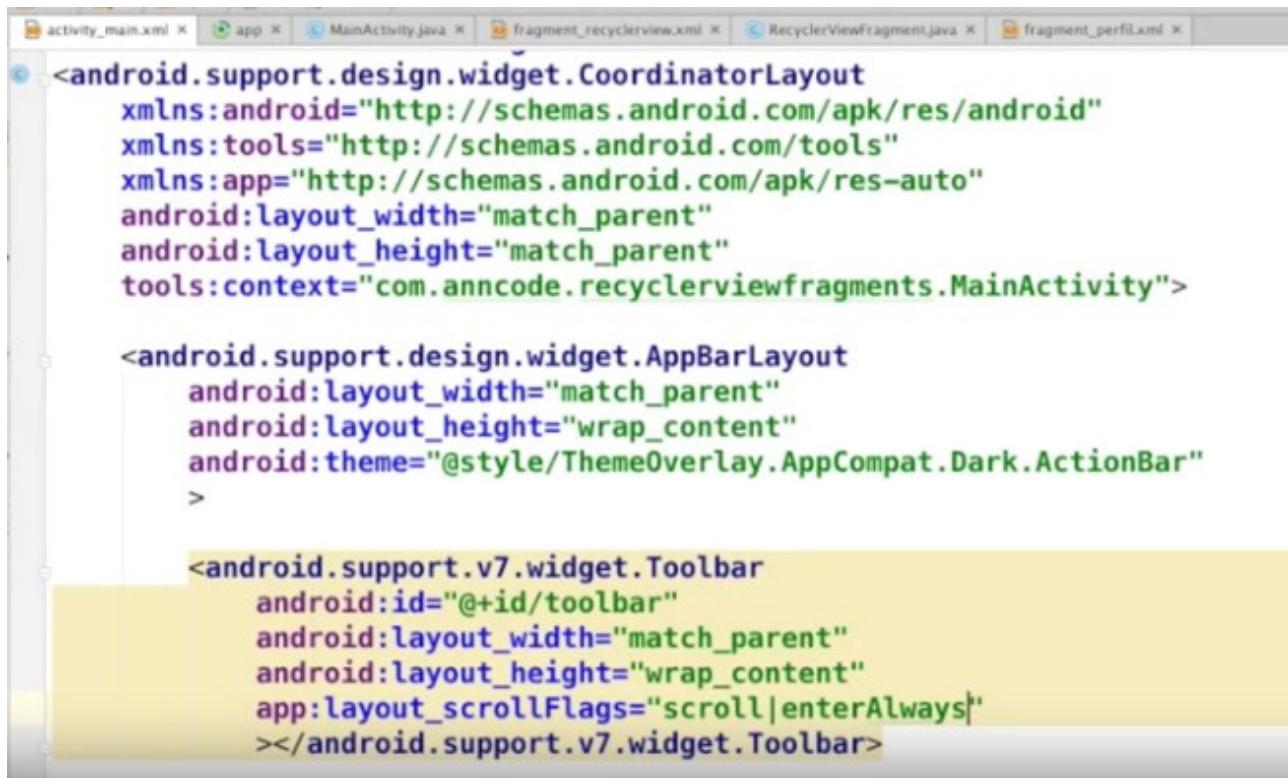


```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.anncode.recyclerviewfragments.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">
        </android.support.design.widget.AppBarLayout>

    </android.support.design.widget.CoordinatorLayout>
```

Vamos a ponerle un id a este toolbar, y vamos a ponerle así toolbar y le vamos a poner por acá una propiedad más. Que precisamente con la etiqueta app:Layout_scrollFlags y lo que va hacer esto, vamos a poner scroll pipe enter Always.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.anncode.recyclerviewfragments.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">
        </android.support.design.widget.AppBarLayout>

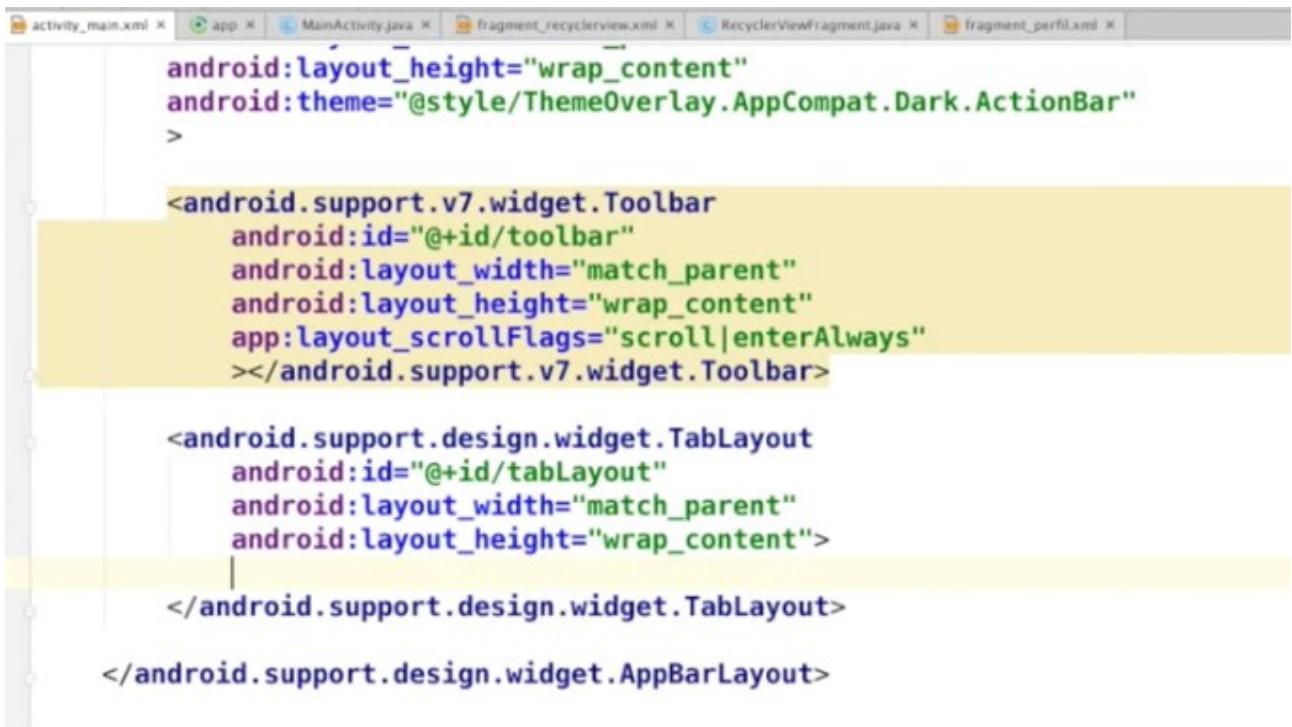
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:layout_scrollFlags="scroll|enterAlways">
        </android.support.v7.widget.Toolbar>
```

Ok, entonces lo que vamos a estar trabajando en nuestro viewPager vamos a tener por aquí un par de tabs.

Vamos a manejar unas tabs aquí, que esas tabs nos van ayudar a precisamente estar manejando el viewPager. Entonces cuando nosotros hagamos esto, cuando nosotros hagamos el scroll, pues vamos a crear un efecto interesante en nuestras tabs. De tal forma que las tabs se van como que a recorrer en la parte superior y va ser lo único que va quedar visible.

Entonces esto lo logramos gracias al coordinator layout, logramos ese efecto. Entonces eso es lo que vamos a colocar, por eso esta instrucción Layout_scrollFlags es lo que nos ayuda a lograr ese efecto en nuestras tabs. Entonces teniendo ya nuestro toolbar, debajo vamos a poner las tabs, precisamente, con la instrucción Tab layout, aquí está.

Vamos a poner width match parent y height wrap content. También le vamos a colocar un id, es importante que todos éstos lleven id porque los vamos a estar ocupando. Los vamos a estar ocupando ahora en nuestro proyecto.



```
activity_main.xml X app X MainActivity.java X fragment_recyclerview.xml X RecyclerViewFragment.java X fragment_perfil.xml X
<android.support.design.widget.AppBarLayout
    android:layout_height="wrap_content"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    >

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_scrollFlags="scroll|enterAlways"
        ></android.support.v7.widget.Toolbar>

    <android.support.design.widget.TabLayout
        android:id="@+id/tabLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        >
        </android.support.design.widget.TabLayout>

</android.support.design.widget.AppBarLayout>
```

Bien, ya tenemos nuestros tabs. Ahora sí, debajo de nuestro appBar, colocaremos nuestro viewPager.

Entonces vamos a buscar nuestro viewPager, aquí está también. support.V4.view.ViewPager, vamos a colocar un match parent. El viewPager va ser como nuestro contenedor, va ser el contenedor donde van a estar viviendo esos fragments.

Ok, entonces si pudiéramos ver u organizar esto un poquito más. Si pudiéramos verlo en forma gráfica, hemos colocado un appBar, le colocamos un toolbar, y debajo del toolbar le hemos colocado un par de tabs. Bueno, solamente hemos colocado el view que va manejar las tabs.

Entonces al toolbar le hemos colocado un efecto de que cuando hagamos scroll, el toolbar desaparezca y solamente queden las tabs.

Y posteriormente después de nuestro appBar viene nuestro ViewPager que va contener nuestros fragments. Vamos a ponerle al ViewPager una propiedad más. App vamos a poner layout_behavior.

Igual Y vamos a colocar aquí @string/appbar_scrolling. Igualmente aquí estamos añadiendo ese efecto. Y cuando lo tengamos te va quedar más claro, cuando tengamos nuestro efecto. Entonces ultimamos un recurso de string/appbar_scrolling_view_behavior y listo. Ya tenemos por ahí nuestro viewPager, y ya está configurado.

```
</></android.support.v7.widget.Toolbar>

<android.support.design.widget.TabLayout
    android:id="@+id/tabLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    </android.support.design.widget.TabLayout>

</android.support.design.widget.AppBarLayout>

<android.support.v4.view.ViewPager
    android:id="@+id/viewPager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">

    </android.support.v4.view.ViewPager>
```

Entonces vamos ahora a nuestro main activity, y en nuestro main activity vamos a estar declarando estos tres views que acabamos de insertar.

Ok, entonces primeramente los voy a declarar de una forma global.

Voy a declarar private toolbar, ok, le voy a poner aquí mi toolbar.

Ahora le voy a poner private.

Private, vamos a poner nuestro TabLayout, aquí está, TabLayout.

Y ahora ponemos nuestro private, nuestro navigation, nuestro viewPager.

Entonces por acá vamos a toolbar, hacer nuestro clásico findViewById(R.id.) Vamos a ponerle toolbar y lo mismo para los otros dos que vamos a manejar nuestros TabLayout. Nuestro TabLayout y también nuestro viewPager.

```
private ArrayList<Contacto> contactos;
private RecyclerView rvContactos;
private Toolbar toolbar;
private TabLayout tabLayout;
private ViewPager viewPager;

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

    toolbar = (Toolbar) findViewById(R.id.toolbar);
    tabLayout = (TabLayout) findViewById(R.id.tabLayout);
    viewPager = (ViewPager) findViewById(R.id.viewPager);
```

Además vamos hacer por acá un validación.

Y después de esto que tenemos por aquí que es el recycler view que esto lo vamos a mover ahora porque nuestro recycler view ya no vive en el main activity. Entonces lo vamos a tener que mover.

Recuerda que el recycler view lo pasamos a nuestro fragment recycler view, Y entonces este código pues bueno debe estar en el archivo de Java de fragment.

Vamos a generar una validación que si de pronto el toolbar está en nulo, entonces consiga el setSupportActionBar de toolbar.

Nos falta trabajar todavía este ViewPager. Todavía no podemos correr nuestro proyecto.

Entonces lo que vamos a hacer para trabajar este ViewPager es que vamos a crear un page adapter.

Ok. Entonces a esas alturas del proyecto, como observas ya tengo ahora demasiadas clases.

Entonces voy a comenzar a organizar un poco más mi proyecto.

Voy a colocar aquí que estoy creando. Vamos a crear una carpeta, no me fijé bien si es una carpeta aquí, un package. Entonces vamos a crear aquí adapter. Entonces aquí en adapter va estar nuestro contacto adapter. Y nada más es lo que va estar allí.

Tenemos además dos fragments. Vamos a crear otra carpeta. Y, esta carpeta se va a llamar fragment. Con f minúscula.

Entonces trata de manejar tus carpetas en minúsculas. Y bueno, vamos a pasar esto por acá. Como que no se deja mucho, vamos a manipularlo. Vamos a darle cut, entonces aquí paste.

Le decimos refactor, sí. Y también el RecyclerView, cut y por acá paste. Esto del refactor es lo que hace la magia para que nos altere nada del código que hemos estado trabajando.

Ok, entonces contacto, detalle contacto. Vamos a crear un package más que se llame Pojo, y entonces esto tiene que estar aquí, contacto tiene que estar en Pojo.

Ya tenemos mejor organizado nuestro proyecto.

Por un lado vamos a tener los adaptadores, que ahora vamos a tener que crear un adaptador, para estar trabajando con el View Pager.

Tenemos, por otro lado, también una carpeta donde estamos concentrando los fragments.

Y, por otro lado, tenemos una carpeta en donde tenemos nuestro Pojo o nuestro modelo de datos.

Bien, entonces vamos a crear nuestro adaptador.

Vamos a crear una nueva clase y se va llamar PageAdapter.

Ese adaptador nos va ayudar a manejar precisamente toda la parte de estar incrustando un fragment en cada tap de mi aplicación.

Ok, entonces aquí en mi PageAdapter, vamos hacer que este PageAdapter herede de la clase FragmentPagerAdapter.

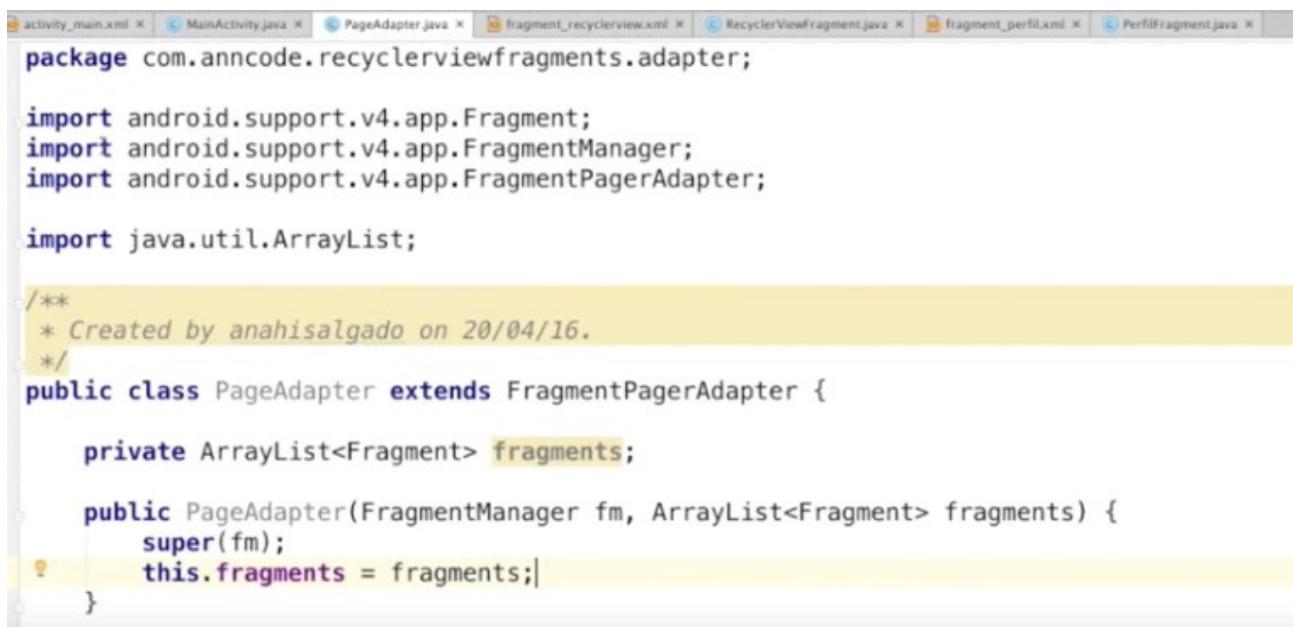
Entonces vamos a poner aquí un alt enter, vamos a implementar aquí los métodos que se nos solicita por ahí. Y además vamos a estar trabajando por aquí. Vamos a dar un alt enter más.

Vamos a crear un constructor y Tenemos you, listo, tenemos estos tres métodos. Además de esto, voy a estar manejando una variable ArrayList Fragment, un arreglo de fragmentos.

Ok, es así como el page adapter, el adaptador, es como va estar viendo los fragments. Cada fragment incrustado en cada tab, va ser un arreglo de fragments. Y así podemos tener.

Y por un momento podríamos tener dos fragments. En este momento tenemos dos fragments o dos tabs. Pero en un futuro podemos tener tres o cuatro. O dinámicamente podemos estar trabajando las tabs de este fragment.

Entonces además aquí del constructor, voy a estar también pasando un ArrayList de fragment.



```
activity_main.xml X MainActivity.java X PageAdapter.java X fragment_recyclerview.xml X RecyclerViewFragment.java X fragment_perfil.xml X PerfilFragment.java X
package com.anncode.recyclerviewfragments.adapter;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

import java.util.ArrayList;

/**
 * Created by anahisalgado on 20/04/16.
 */
public class PageAdapter extends FragmentPagerAdapter {

    private ArrayList<Fragment> fragments;

    public PageAdapter(FragmentManager fm, ArrayList<Fragment> fragments) {
        super(fm);
        this.fragments = fragments;
    }
}
```

Vamos a colocar aquí. Fragments Vamos a estar inicializando esto.

Ok, en nuestro método getItem, lo que vamos a estar devolviendo es precisamente, un fragment en particular.

Entonces vamos a estar manejando a partir de la posición y entonces estar devolviendo un fragment, de uno de los fragments que están incrustados en los tabs.

En nuestro método getCount estaremos devolviendo el tamaño de la lista de fragment. Ok, esto es todo lo que tengo que hacer en mi page adapter.

```
@Override  
public Fragment getItem(int position) {  
    return fragments.get(position);  
}  
  
@Override  
public int getCount() {  
    return fragments.size();  
}
```

Y entonces ahora lo que voy a poner por acá.

Vamos a crear un método, vamos a poner un método.

Puede ser privado o público, void y vamos a poner setUpViewPager. Y este método lo que va hacer es precisamente poner en órbita los fragments.

Poner en órbita los fragments. Y Entonces para ello, primeramente tendré que estar añadiendo los fragments a un arreglo de fragments. Porque recuerda que el page adapter lo que está recibiendo es precisamente nuestro arreglo de fragments.

Entonces lo que nosotros necesitamos aquí recibir es, aquí en SetUpViewPager vamos a estar trabajando con la clase Page Adapter. Primeramente debemos estar añadiendo nuestros fragments.

Entonces vamos a colocar aquí private y vamos a devolver una lista de fragment. Vamos a decirle AgregarFragment.

Entonces aquí vamos a estar manejando la lista que vamos a devolver. Le voy a poner fragments=new ArrayList; y entonces aquí voy a utilizar el método add de la lista fragments.

Y voy a estar añadiendo mi primer fragment. El que quiero que sea RecyclerViewFragment, ese es el primero que quiero mostrar en el primer tab. El segundo, quiero que mostremos el de profile PerfilFragment. Y finalmente devolvemos Devolvemos los fragments.

```
private ArrayList<Fragment> agregarFragments(){  
    ArrayList<Fragment> fragments = new ArrayList<>();  
  
    fragments.add(new RecyclerViewFragment());  
    fragments.add(new PerfilFragment());  
  
    return fragments;  
}
```

Listo, con este método estaremos agregando los fragments a la lista y si ya los tenemos.

Ahora simplemente debemos pasar este arrayList al page adapter. Ok, entonces en el page adapter utilizaré mi viewPager.

Le voy a decir setAdapter y dentro dentro de setAdapter va recibir el PageAdapter que acabamos de crear. Y le vamos a decir getSupportFragmentManager. Y el arreglo, vamos a decir aquí, agregarFragments. Entonces de esta forma ya estamos pasando el soporte de fragment manager.

Y además estamos pasando la lista de fragments que queremos agregar al page adapter, al viewPager.

```
private void setUpViewPager(){
    viewPager.setAdapter(new PageAdapter(getSupportFragmentManager(), agregarFragments()));
}
```

Entonces una vez que tenemos ese viewPager configurado, bueno, hay que agregarlo también al tab layout. Entonces con setUpWithViewPager. Ok, pondremos viewPager y listo.

```
private void setUpViewPager(){
    viewPager.setAdapter(new PageAdapter(getSupportFragmentManager(), agregarFragments()));
    tabLayout.setupWithViewPager(viewPager);
}
```

Entonces nos faltaría nada más llamar nuestro método por acá arriba.

Entonces vamos a llamar nuestro método setUpViewPager.

```
tabLayout = (TabLayout) findViewById(R.id.tabLayout);
viewPager = (ViewPager) findViewById(R.id.viewPager);
setUpViewPager();
```

Por el momento solamente estaremos viendo, podemos probablemente estar haciendo scroller entre los fragments pero todavía no podemos ver la lista de fragments.

Debemos colocar en nuestro archivo manifest, debemos decir que el feature, por aquí debemos colocar windowActionBar a falso. Ok, debe ser por acá en nuestro archivo styles, donde manejamos el estilo del tema.

Entonces vamos a decir aquí. Debajo de éste vamos a colocar un item windowActionBar. Ok, es importante que no coloquemos este que dice Android, ok, y vamos a colocar el false.

Y además vamos a poner el item windowNoTitle. Y le vamos a poner el true.

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
</style>
```

Vamos a regresar el tema que tenemos aquí así. Y lo que he hecho en mi archivo styles es decirle que no tenga un actionBar por defecto, porque al parecer están chocando las actionBar.

Entonces le decimos que no tenga un actionBar por defecto y que tampoco tenga un título.

Entonces de esta forma es como podemos ponerle el appbar. Automáticamente.

Añadiendo RecyclerView en un Fragment

Entonces, una vez que you hemos agregado nuestros dos fragments, ya hemos agregado nuestro elemento ViewPager, pues ahora vamos a añadir en nuestro primer fragment toda la funcionalidad de nuestro RecyclerView.

Recuerdas que nuestro RecyclerView lo pasamos a nuestro fragment_recyclerview, que es el primer fragment que estamos mostrando y que en este momento se está mostrando vacío.

Entonces, vamos a llenar esto y lo que teníamos en nuestro Main Activity, recuerdas por aquí que teníamos toda esta funcionalidad del llenado del RecyclerView.

Entonces, ahora, esta funcionalidad lo que haré será llevármela a mi fragment.

Entonces voy a cortarla, voy a llevármela de aquí.

Y vamos, iremos al archivo que controla este layout. Fragment_recyclerview, aquí está, RecyclerViewFragment y en nuestro método, onCreateView, aquí en nuestro método onCreateView, vamos a declarar, vamos a poner aquí todo lo que tiene que ver con nuestro RecyclerView.

Okey, entonces lo vamos a poner ahí y entonces también vamos a llevarnos por acá esto que tenemos acá.

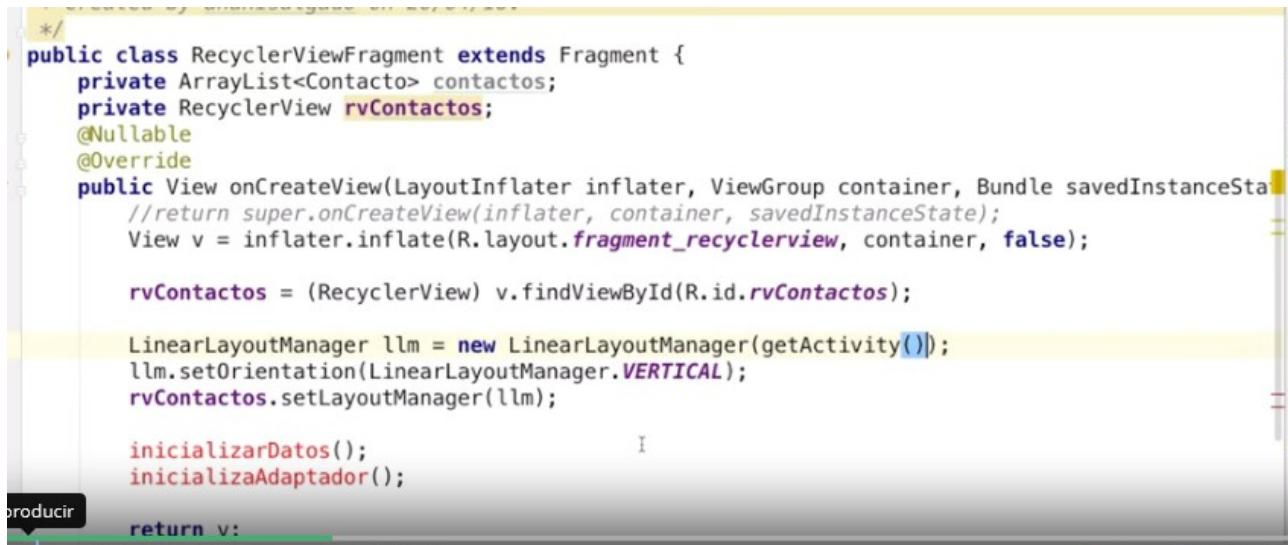
Okey, tanto el ArrayList de contactos como la declaración del view, del RecyclerView. Entonces sí, importamos todo esto.

Y entonces aquí tenemos que hacer algunos ajustes. Este findViewById, en este momento ya no me va a funcionar por sí mismo, entonces voy a utilizar el view.

Recuerda que este view ahora, este view es este layout, entonces donde realmente estamos buscando el RecyclerView es en, precisamente, en este view, fragment_recyclerview. Ahí sí lo vamos a encontrar.

Luego tenemos esto, simplemente tenemos que importarlo con un alt enter.

Aquí debemos colocar, debemos colocar el contexto. Y en el contexto vamos a colocar getActivity.



```
/*
public class RecyclerViewFragment extends Fragment {
    private ArrayList<Contacto> contactos;
    private RecyclerView rvContactos;
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        //return super.onCreateView(inflater, container, savedInstanceState);
        View v = inflater.inflate(R.layout.fragment_recyclerview, container, false);

        rvContactos = (RecyclerView) v.findViewById(R.id.rvContactos);

        LinearLayoutManager llm = new LinearLayoutManager(getActivity());
        llm.setOrientation(LinearLayoutManager.VERTICAL);
        rvContactos.setLayoutManager(llm);

        inicializarDatos();
        inicializaAdaptador();

        producir
        return v;
    }
}
```

Luego tenemos nuestros dos métodos, estos dos métodos los teníamos en nuestro archivo Main Activity, nos los llevaremos, los quitaremos de aquí y los pondremos en nuestro RecyclerViewFragment, ahí. Listo, aquí están los nuestros dos métodos en el ContactoAdaptador, aquí me está diciendo aquí también, vamos a ponerle getActivity.



```
public void inicializarDatos(){
    contactos = new ArrayList<>();
    contactos.add(new Contacto(R.drawable.candy_icon, "Anahi Salgado", "77779999", "anahi@gmai
    contactos.add(new Contacto(R.drawable.yammi_banana_icon, "Pedro Sanchez", "88882222", "ped
    contactos.add(new Contacto(R.drawable.shock_rave_bonus_icon,"Mireya Lopez", "33331111", "m
    contactos.add(new Contacto(R.drawable.forest_mushroom_icon,"Juan Lopez", "44442222", "juan

}

public ContactoAdaptador adaptador;
private void inicializaAdaptador(){
    adaptador = new ContactoAdaptador(contactos, getActivity());
    rvContactos.setAdapter(adaptador);
}
```

La idea es que ahora, pues ya debemos ver en nuestra primera página nuestro RecyclerView.

Okey, entonces lo único que hicimos fue pasar toda la funcionalidad que ya teníamos de Main Activity, donde teníamos ahí nuestro RecyclerView lo pasamos a nuestro fragment.

Tenemos nuestros dos tabs. Vamos a colocarles un pequeño ícono a nuestros tabs. Y para ello, entonces vamos a, vamos a traer, iremos al sitio iconarchive, Icon8, Android icons. Y ahí vamos a buscar, pues primeramente, una estrella. Una estrella donde vamos a estar mostrando pues todos nuestros contactos.

Vamos a buscar uno de person. Okey, person donde, pues realmente lo que tenemos ahí es una lista de contactos.

Vamos a seleccionar uno blanco y lo vamos a descargar.

Vamos a editarlo un poco y vamos a traerlo a nuestro proyecto, Entonces, vamos a utilizar por aquí la opción New, Image Asset y vamos a seleccionar, lo vamos a buscar.

Aquí lo tengo, love. Aquí está nuestro amiguito, vamos a decirle Action Bar and Tab Icons, para que nos cree un ícono, el ícono adecuado. Y viene el nombre del ícono, vamos a decirle contacts. Vamos a darle Next y por aquí ya nos lo va a importar. Okey, vamos a dar Finish.

Entonces, la forma donde vamos a estar colocando esto, vamos a ir a nuestro Main Activity, you tenemos el primer iconito. Y precisamente en el método SetUpViewPager, ahí vamos a colocar en el tabLayout.getTabAt y puedo decirle que en el primer tab, en el tab que tenga el índice 0, le coloque el ícono R.drawable.ic_contacts, y entonces vamos a seleccionar otro ícono para que sea nuestro perfil

```
private void setUpViewPager(){
    viewPager.setAdapter(new PageAdapter(getSupportFragmentManager(), agregar));
    tabLayout.setupWithViewPager(viewPager);

    tabLayout.getTabAt(0).setIcon(R.drawable.ic_contacts);
}
```

Que sea nuestra imagen de perfil. Entonces que nos muestre, bueno, podríamos utilizar este, por ejemplo. Este, ya está aquí.

Y aquí está you nuestra imagen, es este. Vamos a cambiarle nada más el nombre, circled user, circled_user. Lo traemos como lo hicimos con el anterior y seleccionamos aquí la imagen.

Le decimos que sea una imagen que va a ser para tabs, para tabs y le damos a Finish.

Perfecto, you nos lo ha traído y entonces en el siguiente TabLayout.getTabAt, vamos a poner 1.setIcon, R.drawable.

```
private void setUpViewPager(){
    viewPager.setAdapter(new PageAdapter(getSupportFragmentManager(), agregar));
    tabLayout.setupWithViewPager(viewPager);

    tabLayout.getTabAt(0).setIcon(R.drawable.ic_contacts);
    tabLayout.getTabAt(1).setIcon(R.drawable.ic_action_name);
}
```

Vamos a ver cómo se ve. Sí, es el de action_name. Bueno, podemos colocarle el nombre que querramos. Bien, vamos a correrlo y lo que ahora debemos ver es pues, debemos ver nuestras imágenes en cada tab.



Añadiendo un Fragment desde un layout

Además de añadir un fragment programáticamente como lo vimos en el video anterior, también podemos añadir un fragment a través de un layout.

Como podemos ver en el siguiente código, estamos añadiendo un fragment a través de la etiqueta fragment, como si el fragment se comportará como un view más.

Entonces es muy importante que definamos los atributos obligatorios, el width, el height de cada fragment.

```
<fragment
    android:id="@+id/frgLista"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.anncode.correos.Lista"
    android:layout_weight="6"
    ></fragment>
```

```
<fragment
    android:id="@+id/frgDetalle"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.anncode.correos.Detalles"
    android:layout_weight="4"
    ></fragment>
```

Y además una propiedad importante, la propiedad android name.

Como observas, la propiedad android name, tiene asignado una clase, Una clase que será la clase que herede de la clase fragment, de la clase padre fragment.

Esta clase que está indicada con este atributo android name, nos está indicando que será la clase que tenga todo el control de esta etiqueta view. Es decir, de este fragment.

Ésta es una forma también, en donde podemos agregar fragments a través de un layout.