

PMDM05.- Interacciones entre clases y actividades.

Caso práctico



Ron Lach (Pexels)

El equipo ha realizado los primeros diseños, mediante los cuales han adquirido muchos conocimientos.

- Una necesidad que me surge ahora -indica María-, es poder utilizar una nueva pantalla en la que se pueda solicitar al usuario alguna opción, fuera de la pantalla principal (activity_main) de la app.
- Bien visto -responde Ada-. Hasta ahora todas las apps desarrolladas han sido muy sencillas, sólo se han presentado diseños aislados de interfaces, es decir, no existen varias pantallas que interactúen entre sí o se pasen datos de unas a otras.
- ¡Claro! -exclama Juan-. Las aplicaciones constan de varias pantallas entrelazadas entre ellas.
- Ha llegado el momento de avanzar -explica Ada-. Vais a aprender a utilizar Intents, Splash Screen y los menús habituales de las aplicaciones de dispositivos para establecer la configuración de la aplicación, o para acceder a distintas opciones que no tiene sentido que se engloben en la actividad principal de la aplicación.



[Ministerio de Educación y Formación Profesional](#). (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#) 

1.- Intents.

Caso práctico

Los Intents permiten que las pantallas interactúen entre sí, e incluso con otras aplicaciones.

- Os propongo un nuevo reto -explica Ada-. La app ha de ser capaz de abrir un archivo pdf.
- ¿Una pista? -solicita Juan.
- Para conseguirlo -prosigue Ada-, la app deberá encontrar una app en el dispositivo que sea capaz de abrir este tipo de archivos.

Esta y otras opciones de interacción se realizan gracias al objeto Intent, que podrá ser llamado de forma implícita o explícita.



Mikhail ([Pexels](#))

Una **Intent** es un objeto de acción que usamos para solicitar una acción de otro componente de la aplicación. Aunque los intents facilitan la comunicación entre los componentes de muchas maneras, existen tres casos de uso fundamentales:

✓ Para comenzar una actividad:

Una **Activity** representa una única pantalla en una aplicación. Puedes iniciar una nueva instancia de una **Activity** pasando una Intent a **startActivity()**. La **Intent** describe la actividad que se debe iniciar y contiene los datos necesarios para ello.

Si deseas recibir un resultado de la actividad cuando finalice, llama a **startActivityForResult()**. La actividad recibe el resultado como un objeto **Intent** separado en el callback de **onActivityResult()** de la actividad.

✓ Para iniciar un servicio:

Un **Service** es un componente que realiza operaciones en segundo plano sin una interfaz de usuario. Puede iniciar un servicio para realizar una operación única (como descargar un archivo) pasando una **Intent** a **startService()**. La **Intent** describe el servicio que se debe iniciar y contiene los datos necesarios para ello.

✓ **Para entregar un mensaje:**

Un mensaje es un aviso que cualquier aplicación puede recibir. El sistema entrega varios mensajes de eventos del sistema, como cuando el sistema arranca o el dispositivo comienza a cargarse. Puedes enviar un mensaje a otras apps pasando una **Intent** a **sendBroadcast()**, **sendOrderedBroadcast()** o **sendStickyBroadcast()**.

1.1.- Tipos de Intents.

Existen dos tipos de intents:

- ✓ **Intents explícitas**: especifican qué componente se debe iniciar mediante su nombre (el nombre de clase completamente calificado). Usualmente, el usuario usa una intent explícita para iniciar un componente en su propia aplicación porque conoce el nombre de clase de la actividad o el servicio que desea iniciar. Por ejemplo, puede utilizarla para iniciar una actividad nueva en respuesta a una acción del usuario o iniciar un servicio para descargar un archivo en segundo plano.

Cuando crea una **intent explícita** para iniciar una actividad o un servicio, la aplicación inicia inmediatamente el componente de la aplicación especificado en el objeto Intent.

- ✓ **Intents implícitas**: no se nombra el componente específicamente; pero, en cambio, se declara una acción general para realizar, lo que permite que un componente de otra aplicación la maneje. Por ejemplo, si desea mostrar al usuario una ubicación en un mapa, puede usar una intent implícita para solicitar que otra aplicación capaz muestre una ubicación específica en un mapa.

Cuando crea una **intent implícita**, el sistema Android busca el componente apropiado para iniciar comparando el contenido de la intent con los **filtros de intents** declarados en el archivo de manifiesto (**AndroidManifest.xml**) de otras aplicaciones en el dispositivo. Si la intent coincide con un filtro de intents, el sistema inicia ese componente y le entrega el objeto **Intent**. Si varios filtros de intents son compatibles, el sistema muestra un cuadro de diálogo para que el usuario pueda elegir la aplicación que se debe usar.

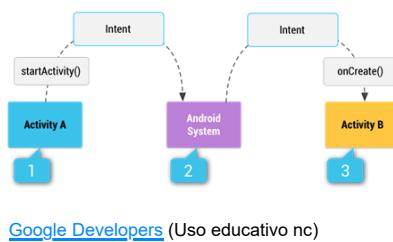


Figura 1: Ilustración de la forma en que se entrega una intent implícita mediante el sistema para iniciar otra actividad. [1] La **actividad A** crea una **Intent** con una descripción de acción y la pasa a **startActivity()**. [2] El sistema Android busca en todas las apps un filtro de intents que coincide con la intent. Cuando se encuentra una coincidencia, [3] el sistema inicia la actividad coincidente (**actividad B**) invocando su método **onCreate()** y pasándole a la **Intent**. Fuente: Android Developer.

Un **filtro de intents** es una expresión en el archivo de manifiesto de una aplicación que especifica el tipo de intents que el componente podría recibir. Por ejemplo, declarando un filtro de intents para una actividad, permite que otras aplicaciones inicien directamente la actividad con cierto tipo de intent. Así mismo, si *no* declara ningún filtro de intent para una actividad, solo se la puede iniciar con una intent explícita.

1.2.- Creación de una intent.

Un objeto **Intent** tiene información que el sistema Android usa para determinar qué componente debe iniciar (como el nombre exacto del componente o la categoría del componente que debe recibir la intent), además de información que el componente receptor usa para realizar la acción correctamente (por ejemplo, la acción que debe efectuar y los datos en los que debe actuar).

La información principal que contiene una **Intent** es la siguiente:

Nombre del componente, acción, datos, categoría, extras e indicadores.

De toda esa posible información, tendrá especial interés el paso de extras, que consiste en pares de valores clave que tienen información adicional necesaria para lograr la acción solicitada. Al igual que algunas acciones usan tipos particulares de URI de los datos, algunas acciones también usan extras particulares.

Podemos agregar datos de campos extra con varios métodos **putExtra()**, cada uno de los cuales acepta dos parámetros: el nombre clave y el valor. También puedes crear un objeto **Bundle** con todos los datos de campos extra y luego insertar el **Bundle** en el Intent con **putExtras()**.

1.3.- Ejemplo de una intent explícita.

Una intent explícita es una intent que se usa para iniciar un componente específico de la aplicación, como una actividad o un servicio particular en la aplicación. Para crear una intent explícita, define el nombre de componente para el objeto **Intent**, todas las otras propiedades de la intent son opcionales.

Ejemplo

Si creaste un servicio en tu app denominado **DownloadService**, diseñado para descargar un archivo de la Web, puedes iniciararlo con el siguiente código:

```
1 // Executed in an Activity, so 'this' is the Context
2
3 // The fileUrl is a string URL, such as "http://www.example.com/image.png"
4
5 Intent downloadIntent = new Intent(this, DownloadService.class);
6
7 downloadIntent.setData(Uri.parse(fileUrl));
8
9 startService(downloadIntent);
```

El constructor **Intent(Context, Class)** proporciona **Context** a la aplicación; y el componente, un objeto **Class**. De esta manera, la intent inicia explícitamente la clase **DownloadService** en la app.

1.4.- Ejercicio resuelto 1.

Ejercicio Resuelto

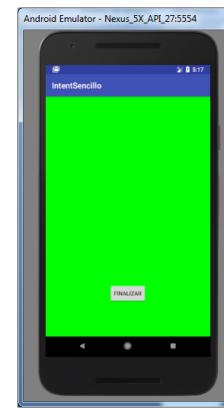
Crear una aplicación con 2 ventanas. la primera tendrá un fondo rojo y un botón que al ser clicado me lleve a la segunda ventana. La segunda ventana tendrá fondo verde y un botón que al ser clicado, hará que finalice la aplicación.

[Mostrar retroalimentación](#)

Tendremos que crear 2 actividades y vincularlas entre sí.



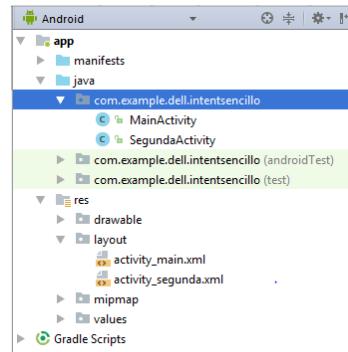
[Google Developers](#) (Uso educativo nc)



[Google Developers](#) (Uso educativo nc)

Para crear una nueva Activity **File->New->Activity->Empty Activity**

Quedará tal y como se muestra en la captura de Android.



[Google Developers](#) (Uso educativo nc)

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     android:background="#FF0000"
9     tools:context="com.example.dell.intentssencillo.MainActivity">
10
11     <Button
12         android:id="@+id/btnIrSegundaActividad"
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:layout_marginTop="32dp"
16         android:text="Ir a segunda ventana"
17         app:layout_constraintEnd_toEndOf="parent"
18         app:layout_constraintLeft_toLeftOf="parent"
19         app:layout_constraintRight_toRightOf="parent"
```

```
20     app:layout_constraintStart_toStartOf="parent"
21     app:layout_constraintTop_toTopOf="parent" />
22 </android.support.constraint.ConstraintLayout>
```

activity_segunda.xml

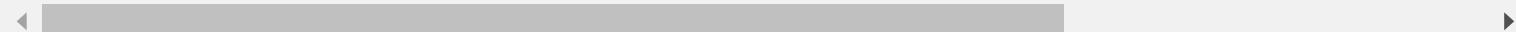
```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      android:background="#00FF00"
9      tools:context="com.example.dell.intentsencillo.SegundaActivity">
10     <Button
11         android:id="@+id/btnFinalizar"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:layout_marginBottom="84dp"
15         android:text="Finalizar"
16         app:layout_constraintBottom_toBottomOf="parent"
17         app:layout_constraintLeft_toLeftOf="parent"
18         app:layout_constraintRight_toRightOf="parent" />
19 </android.support.constraint.ConstraintLayout>
```

MainActivity.java

```
1 package com.example.dell.intentsencillo;
2 import android.content.Intent;
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7
8 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
9     Button btnIrSegundaActividad;
10    @Override
11    protected void onCreate(Bundle savedInstanceState) {
12        super.onCreate(savedInstanceState);
13        setContentView(R.layout.activity_main);
14        btnIrSegundaActividad=(Button)findViewById(R.id.btnIrSegundaActividad);
15        btnIrSegundaActividad.setOnClickListener(this);
16    }
17
18    @Override
19
20    public void onClick(View v) {
21        Intent i=new Intent(getApplicationContext(), SegundaActivity.class);
22        startActivity(i);
23    }
24 }
```

SegundaActivity.java

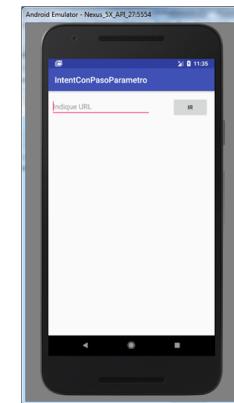
```
1 package com.example.dell.intentsencillo;
2
3 import android.os.Build;
4 import android.support.v7.app.AppCompatActivity;
5 import android.os.Bundle;
6 import android.view.View;
7 import android.widget.Button;
8
9 public class SegundaActivity extends AppCompatActivity {
10     Button btnFinalizar;
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_segunda);
15         btnFinalizar=(Button)findViewById(R.id.btnFinalizar);
16         btnFinalizar.setOnClickListener(new View.OnClickListener() {
17             @Override
18             public void onClick(View v) {
19                 //Para poder salir completamente de la aplicación no vale finish() ya que dicho método cerrar
20                 //Utilizaremos finishAffinity(), pero esta solo es válida a partir de la API 16
21                 if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
22                     finishAffinity();
23                 }
24             }
25         });
26     }
27 }
28 }
```



1.5.- Ejercicio resuelto 2.

Ejercicio Resuelto

Crea una aplicación que tenga un **intent** que cargue en una segunda actividad una página web a través de una **WebView**, de forma que se especifique la URL a cargar en la primera actividad.



[Google Developers \(Uso educativo nc\)](#)

[Mostrar retroalimentación](#)

activity_main.xml

```
1 | <?xml version="1.0" encoding="utf-8"?>
2 | <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
3  xmlns:app="http://schemas.android.com/apk/res-auto"
4  xmlns:tools="http://schemas.android.com/tools"
5  android:layout_width="match_parent"
6  android:layout_height="match_parent"
7  tools:context="com.example.dell.intentconpasoparametro.MainActivity">
8
9  <EditText
10    android:id="@+id/etURL"
11    android:layout_width="245dp"
12    android:layout_height="wrap_content"
13    android:layout_marginStart="8dp"
14    android:layout_marginTop="16dp"
15    android:hint="Indique URL"
16    app:layout_constraintStart_toStartOf="parent"
17    app:layout_constraintTop_toTopOf="parent"
18    tools:ignore="RtlCompat" />
19
20 <Button
21    android:id="@+id/btnIr"
22    android:layout_width="wrap_content"
23    android:layout_height="wrap_content"
24    android:layout_marginRight="16dp"
25    android:layout_marginTop="16dp"
26    android:text="Ir"
27    app:layout_constraintRight_toRightOf="parent"
28    app:layout_constraintTop_toTopOf="parent" />
29
30 </android.support.constraint.ConstraintLayout>
```

activity_segunda.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context="com.example.dell.intentconpasoparametro.SegundaActivity">
8
9     <WebView
10         android:id="@+id/wv"
11         android:layout_width="match_parent"
12         android:layout_height="400dp"
13         android:layout_marginBottom="60dp"
14         android:layout_marginTop="24dp"
15         app:layout_constraintBottom_toBottomOf="parent"
16         app:layout_constraintHorizontal_bias="0.0"
17         app:layout_constraintLeft_toLeftOf="parent"
18         app:layout_constraintRight_toRightOf="parent"
19         app:layout_constraintTop_toTopOf="parent"
20         app:layout_constraintVertical_bias="0.877"></WebView>
21
22 </android.support.constraint.ConstraintLayout>
```

mainActivity.java

```
1 package com.example.dell.intentconpasoparametro;
2
3 import android.content.Intent;
```

```
4 import android.support.v7.app.AppCompatActivity;
5 import android.os.Bundle;
6 import android.view.View;
7 import android.widget.Button;
8 import android.widget.EditText;
9
10 public class MainActivity extends AppCompatActivity {
11     EditText etURL;
12     Button btnIr;
13
14     @Override
15
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19         etURL=(EditText)findViewById(R.id.etURL);
20         btnIr=(Button)findViewById(R.id.btnIr);
21         btnIr.setOnClickListener(new View.OnClickListener() {
22
23             @Override
24
25             public void onClick(View v) {
26                 Intent i=new Intent(getApplicationContext(), SegundaActivity.class);
27                 i.putExtra("direccion", etURL.getText().toString());
28                 startActivity(i);
29             }
30         });
31     }
32 }
33
34 }
```

SegundaActivity.java

```
1 package com.example.dell.intentconpasoparametro;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.webkit.WebView;
7 import android.widget.Button;
8
9 public class SegundaActivity extends AppCompatActivity {
10     WebView wv;
11
12     @Override
13
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_segunda);
17         wv=(WebView)findViewById(R.id.wv);
18         Bundle bundle=getIntent().getExtras();
19         String url=bundle.getString("direccion");
20         wv.loadUrl("http://" + url);
21     }
22 }
```

AndroidManifest.xml

Además, debemos incorporar en **AndroidManifest.xml** permiso de acceso a Internet.

```
1 | <uses-permission android:name="android.permission.INTERNET"/>
```



2.- Splash screen.

Caso práctico



Liza Summer ([Pexels](#))

- Enhorabuena -felicita Ada al equipo-. Habéis conseguido resolver el primer reto. Ahora os propongo un segundo reto.
- ¡Genial! -exclama María.
- Debéis realizar una app -explica Ada-, que muestre a los usuarios un mensaje cada vez que sea lanzada.
- ¿Cuál será el elemento nuevo a utilizar? -pregunta Pedro.
- En esta ocasión es el tipo de ventana denominada **Splash Screen** -responde Ada-, que permite a las apps mostrar información relevante cuando estas son lanzadas.

Es habitual que muchas aplicaciones incorporen una pantalla de presentación al iniciarse la aplicación, y que se muestra durante apenas unas decimas de segundo, o bien pocos segundos. A dicha pantalla se le suele llamar **Splash Screen**, y el objetivo es mejorar la experiencia de usuario, a la vez que proporcionar información adicional de esponsorización, patrocinio o información acerca de la empresa/s desarrolladora/s o colaboradora/s o bien, simplemente información comercial o publicitaria.

2.1.- Ejercicio Resuelto 3.

Ejercicio Resuelto

Construye una app Android que muestre una pantalla cuando se lance con una imagen que ocupe toda la pantalla del dispositivo. Al cerrarse se mostrará la pantalla por defecto de la app.

[Mostrar retroalimentación](#)

Lo primero que haremos será preparar la imagen que utilizaremos. Lo ideal, al trabajar con imágenes en Android, será proporcionar diferentes resoluciones para que puedan adaptarse a los diferentes tamaños de pantallas disponibles en los dispositivos Android que usemos. En nuestro caso la llamaremos "android.png".

Tras buscar y adaptar la imagen que queremos que se muestre, definiremos una nueva Actividad que nosotros llamaremos *SplashScreenActivity* y que llevará asociado un fichero **activity_splash_screen.xml**. Dentro de este fichero .XML, incorporaremos un control **ImageView**, ocupando todo el *layout* mediante el uso del atributo **android:scaleType** con valor **fitXY**.



Archivo de manifiesto

Un archivo ordinario de manifiesto podría mostrar este aspecto donde se declaran las dos actividades y se arranca la principal.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.dell.tarea212">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="@string/app_name"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:theme="@style/AppTheme">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15                 <category android:name="android.intent.category.LAUNCHER" />
16             </intent-filter>
17         </activity>
18
19         <activity android:name=".SplashScreenActivity"></activity>
20
21     </application>
22 </manifest>
```

Podemos apreciar como la actividad que tiene la etiqueta hija relacionada con la carga es MainActivity:

```
1 | <category:android:name="android.intent.category.LAUNCHER" />
```

Esto significa que por defecto cargará la actividad principal. Nosotros lo cambiaremos para que inicialmente se cargue la actividad del splash screen, moviendo esta línea a la actividad SplashScreen.

Archivo de manifiesto

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3   package="com.example.dell.tarea212">
4
5   <application
6     android:allowBackup="true"
7     android:icon="@mipmap/ic_launcher"
8     android:label="@string/app_name"
9     android:roundIcon="@mipmap/ic_launcher_round"
10    android:supportsRtl="true"
11    android:theme="@style/AppTheme">
12      <activity android:name=".MainActivity"></activity>
13
14      <activity android:name=".SplashScreenActivity">
15        <intent-filter>
16          <action android:name="android.intent.action.MAIN" />
17          <category android:name="android.intent.category.LAUNCHER" />
18        </intent-filter>
19      </activity>
20
21    </application>
22 </manifest>
```

SplashScreenActivity.java

Código java del splash screen.

```
1 package com.example.dell.tarea53;
2
3 import android.content.Intent;
4 import android.content.pm.ActivityInfo;
5 import android.support.v7.app.AppCompatActivity;
6 import android.os.Bundle;
7 import android.view.Window;
8 import java.util.Timer;
9 import java.util.TimerTask;
10
11
12 public class SplashScreenActivity extends AppCompatActivity {
13
14     @Override
15
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         // Establecemos orientación exclusiva de retrato
19         setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
20         // Ocultamos la barra de título
21         getSupportActionBar().hide();
22         setContentView(R.layout.activity_splash_screen);
23
24         TimerTask task = new TimerTask() {
25
26             @Override
27
28             public void run() {
29                 // Arrancamos la siguiente actividad
30                 Intent mainIntent = new Intent().setClass(
31                     SplashScreenActivity.this, MainActivity.class);
32                 startActivity(mainIntent);
33                 // Cerramos esta actividad para que el usuario no pueda volver a ella mediante botón de volver
34                 finish();
35             }
36         };
37
38
39         // Simulamos un tiempo en el proceso de carga durante el cual mostramos el splash screen
```

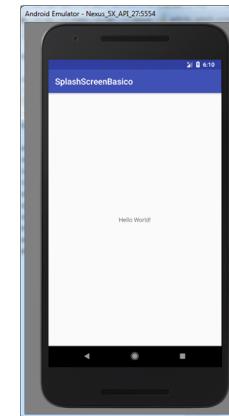
```
40     Timer timer = new Timer();
41     timer.schedule(task, 2000); //Tiempo de espera del temporizador en milisegundos
42 }
43 }
```



En cuanto al **MainActivity.java** y el **activity_main.xml**, no haremos nada en particular, dejando el clásico *Hola Mundo!*. Por ello, tras los dos segundos de retardo, aparecerá la pantalla con el texto Hola Mundo!.



[Google Developers](#) (Uso educativo nc)



[Google Developers](#) (Uso educativo nc)

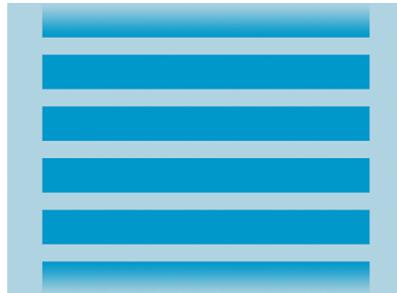
SplashScreenActivity.java

Una forma alternativa de programar el **SplashScreenActivity.java**, ayudándonos de los asistentes.

```
1 package com.example.dell.splashscreen;
2
3 import android.content.Intent;
4 import android.os.Handler;
5 import android.support.v7.app.AppCompatActivity;
6 import android.os.Bundle;
7
8 public class SplashScreenActivity extends AppCompatActivity {
9
10    @Override
11
12    protected void onCreate(Bundle savedInstanceState) {
13        super.onCreate(savedInstanceState);
14        setContentView(R.layout.activity_splash_screen);
15
16        new Handler().postDelayed(new Runnable() {
17
18            @Override
19
20            public void run() {
21                //Lanzaremos despues de 3000ms(3s), el MainActivity
22                Intent intent=new Intent(SplashScreenActivity.this, MainActivity.class);
23                startActivity(intent);
24            }
25        }, 3000);
26    }
27}
```

3.- ListView.

Caso práctico



- Vamos a por el tercer reto -informa Ada.
- ¿De qué se trata esta vez? -pregunta María.
- Ahora debéis incorporar listas a la aplicación -explica Ada-. Para ello utilizaréis el elemento **ListView**.
- ¡Es verdad! -exclama María-. Las listas se usan en las pantallas que muestran los contactos de una agenda, los mensajes de un chat, etc. La longitud y el número de elementos a mostrar es algo dinámico y cambia en cada instante dependiendo de la situación concreta.
- En vuestro caso -sigue explicando Ada-, las aplicaciones deben mostrar una lista de elementos conseguidos a partir de una consulta a distintos archivos que se encuentran en el dispositivo, como imágenes o documentos PDFs. Dependiendo del número de elementos disponibles en el dispositivo se mostrará un número de elementos u otros.

Un **ListView** es un grupo de vistas que muestra una lista de elementos desplazables. Los elementos de la lista se insertan automáticamente en la lista con un adaptador (clase **Adapter**) que toma contenido de una fuente, como una matriz o consulta de base de datos, y convierte cada resultado en una vista que se dispone en la lista.

Nos encontramos varias posibilidades, dependiendo de la complejidad que queramos atribuir a nuestro **ListView**. Concretamente, podremos usar un **ListView** con un **adaptador de tipo String** (es el caso más simple), o bien usar un **ListView** con un **adaptador personalizado** (caso más complejo y que trataremos en un apartado posterior).

Para insertar vistas de forma dinámica con un adaptador nos ayudaremos de la clase **AdapterView**. Y dentro de esta (extienden de ella), podremos usar la clase **ListView**, o también la clase **GridView**, de forma que enlazaremos la instancia **AdapterView** con un **Adapter**, que recupera datos de una fuente externa y crea una **View** que representa cada entrada de datos.

Android proporciona varias subclases de **Adapter** que resultan útiles para recuperar diferentes tipos de datos y generar vistas para una **AdapterView**. Los dos adaptadores más comunes son los siguientes:

- ✓ ArrayAdapter
- ✓ SimpleCursorAdapter

En particular nos centraremos en **ArrayAdapter**.

3.1.- ArrayAdapter.

Usaremos este adaptador cuando la fuente de datos sea una matriz. Según la configuración predeterminada, **ArrayAdapter** crearemos una vista para cada elemento de la matriz llamando a **toString()** en cada elemento y disponiendo los contenidos en una **TextView**.

Por ejemplo, si hay una matriz de strings que deseamos visualizar en una **ListView**, inicializaremos un nuevo **ArrayAdapter** usando un constructor para especificar el diseño de cada string y la matriz de strings:

```
1 | ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, myStringArray);
```

Los argumentos para este constructor son los siguientes:

- ✓ El **Context** de tu app.
- ✓ El diseño que contiene un **TextView** para cada string de la matriz.
- ✓ La matriz de strings.

Luego, simplemente llamaremos a **setAdapter()** en nuestro **ListView**:

```
1 | ListView listView = (ListView) findViewById(R.id.listView);
2 | listView.setAdapter(adapter);
```

Para personalizar el aspecto de cada elemento, podemos anular el método **toString()** de los objetos de nuestra matriz. Como alternativa, si deseamos crear una vista para cada elemento que no sea una **TextView** (por ejemplo, si deseamos una **ImageView** para cada elemento de la matriz), extenderemos la clase de **ArrayAdapter** y anularemos **getView()** a fin de mostrar el tipo de vista que queramos para cada elemento.

Si durante el ciclo de vida de nuestra aplicación cambiamos los datos subyacentes que lee nuestro adaptador, debemos llamar a **notifyDataSetChanged()**. Esto le notificará a la vista anexada que se modificaron los datos y que debe actualizarse.

3.2.- Manejo de eventos de clicado.

Podemos atender a eventos de clicado en cada elemento de una **AdapterView** al implementar la interfaz **AdapterView.OnItemClickListener**.

Ejemplo

```
1 // Create a message handling object as an anonymous class.  
2  
3 private OnItemClickListener mMessageClickedHandler = new OnItemClickListener() {  
4  
5     public void onItemClick(AdapterView parent, View v, int position, long id) {  
6         // Do something in response to the click  
7     }  
8  
9 };  
10  
11 listView.setOnItemClickListener(mMessageClickedHandler);
```

3.3.- Ejemplo de ListView con adaptador de tipo String. Ejercicio resuelto 4.

Ejercicio Resuelto

En este caso, vamos a tratar de crear un ListView sencillo, para lo cual trataremos de que lo que se muestre en cada ítem, sea simplemente una cadena de texto. Además dotaremos a cada ítem del ListView de la capacidad de responder al evento de clicado. Concretamente, nuestra aplicación será una sencilla agenda de contactos en la cual, al clicar en un contacto, se lanzará un Toast mostrando el número de teléfono asociado. El aspecto será el siguiente:

Pantalla inicial:



[Google Developers \(Uso educativo nc\)](#)

Tras clicar en Juanmi:



[Google Developers \(Uso educativo nc\)](#)

[Mostrar retroalimentación](#)

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     android:orientation="vertical"
9     tools:context="com.example.dell.listviewdestringsencillo.MainActivity">
10
11     <TextView
12         android:id="@+id/tv1"
13         android:layout_width="match_parent"
14         android:layout_height="wrap_content"
15         android:text="Listado de contactos"
16         android:textSize="20dp"
17         android:textColor="#DD6633"
18         android:gravity="center"/>
19
20     <ListView
21         android:id="@+id/lv1"
22         android:layout_width="match_parent"
23         android:layout_height="match_parent"></ListView>
24
25 </LinearLayout>
```

MainActivity.java

```
1 package com.example.dell.listviewdestringsencillo;
2
3 import android.app.AlertDialog;
4 import android.app.Dialog;
5 import android.content.DialogInterface;
6 import android.support.v7.app.AppCompatActivity;
7 import android.os.Bundle;
8 import android.view.View;
9 import android.widget.AdapterView;
10 import android.widget.ArrayAdapter;
11 import android.widget.ListView;
12 import android.widget.TextView;
13 import android.widget.Toast;
14
15 public class MainActivity extends AppCompatActivity {
16     private String[] nombres = {"Pedro", "Ana", "Rosa", "Mario", "Pepe", "Juanmi", "María Luisa", "Chispi"};
17     private String[] telefonos = { "658231234", "623123400", "912334489", "918732312", "657762598", "66312004
18     private ListView lv1;
19     private int posicion;
20
21     @Override
22
23     protected void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         setContentView(R.layout.activity_main);
26         lv1 =(ListView)findViewById(R.id.lv1);
27         ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, nom
28         lv1.setAdapter(adapter);
29         lv1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
30
31             @Override
32
33             public void onItemClick(AdapterView adapterView, View view, int i, long l) {
34                 posicion=i;
35                 Toast.makeText(MainActivity.this, "El telefono de " + nombres[i] + " es " + telefonos[i], Toa
36                 }
37             });
38         }
39     }
```



3.4.- ListView con adaptadores personalizados.

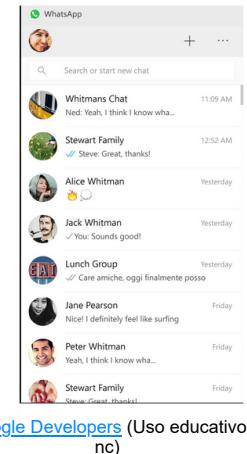
El caso más habitual de uso de un **ListView** es aquel en el cual los ítems que forman parte de dicho **ListView** sean un conjunto de componentes o vistas. Por ejemplo, en aplicaciones como la agenda telefónica de nuestro móvil, o como *Whatsapp*, se presentan unos listados (en realidad suelen estar desarrollados con una clase **RecyclerView**, que es una evolución de **ListView**), donde cada ítem, presenta varios componentes:

En la imagen anterior podemos ver que cada ítem del listado, a su vez está compuesto por una imagen, un nombre del chat, el último mensaje escrito y la fecha de ultima actualización del chat.

Esto pone de manifiesto que al final se hace necesario tratar de incorporar estructuras complejas que serán manejadas mejor si hacemos definiciones de clase que contengan todos estos componentes. Además, el ítem del **ListView** que queramos manejar, deberá ser capaz de escuchar eventos que sucedan sobre ellos, ya sean eventos sobre el ítem global (por ejemplo, para la imagen anterior, al clicar en el ítem, se abre una nueva **Activity** que me lleva al chat específico), o bien eventos sobre un componente del ítem (por ejemplo, para la imagen anterior, responder al clicado sobre la imagen para poder verla en grande en una nueva **Activity**).

Será necesario desarrollar un adaptador personalizado. Dicho adaptador se podrá crear dentro de la actividad principal, pero es más frecuente crearlo en un fichero específico aparte. Dicha clase extenderá de la clase **BaseAdapter** y dentro de ella definiremos un objeto de la clase **Context**, así como alguna estructura dinámica para contener cada uno de los ítems, normalmente un objeto **List** o **ArrayList**. Dicha estructura dinámica será una colección de objetos que se ajusten a nuestro universo. Por ejemplo, podremos crear un **ArrayList** de objetos de tipo **Contacto** para crear una agenda de teléfono.

Al extender de la clase **BaseAdapter**, Android Studio nos forzará a sobrescribir los métodos **getCount()**, **getItem()**, **getItemId()** y **getView()**. De todos estos, sin duda el más importante será **getView()**, y con él configuraremos la apariencia o vista de cada ítem que tenga mi **ListView**. En el ejercicio resuelto siguiente se aclarará esto.

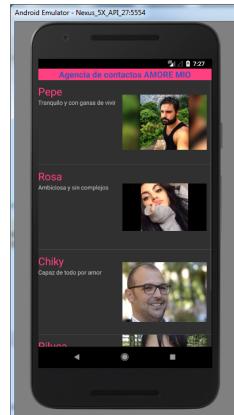


3.4.1.- Ejercicio resuelto 5.

Ejercicio Resuelto

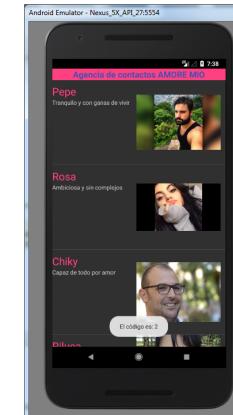
Vamos a crear una especie de agenda de contactos personales.

El aspecto que tendrá la aplicación será:



[Google Developers \(Uso educativo nc\)](#)

Al clicar en Rosa, me mostrará su código:



[Google Developers \(Uso educativo nc\)](#)

[Mostrar retroalimentación](#)

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical">
8
9     <TextView
10         android:layout_width="match_parent"
11         android:layout_height="wrap_content"
12         android:text="Agencia de contactos AMORE MIO"
13         android:background="@color/colorAccent"
14         android:textColor="@color/colorPrimary"
15         android:textSize="20dp"
16         android:textStyle="bold"
17         android:gravity="center"/>
18
19     <ListView
20         android:id="@+id/lv"
21         android:layout_width="match_parent"
22         android:layout_height="match_parent">
23     </ListView>
24
25 </LinearLayout>
```

item.xml

Creamos un fichero XML adicional (**item.xml**) que contiene la distribución de cada ítem que aparecerá en el ListView

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="horizontal"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent">
6
7     <LinearLayout
8         android:layout_width="200dp"
9         android:layout_height="200dp"
10        android:orientation="vertical">
11
12         <TextView
13             android:id="@+id/tvNombre"
14             android:layout_width="match_parent"
15             android:layout_height="wrap_content"
16             android:textSize="25dp"
17             android:textColor="@color/colorAccent"
18             android:layout_marginTop="10dp"/>
19
20         <TextView
21             android:id="@+id/tvDescripcion"
22             android:layout_width="match_parent"
23             android:layout_height="wrap_content"
24             android:layout_marginBottom="10dp"/>
25
26     </LinearLayout>
27
28     <ImageView
29         android:id="@+id/imgFoto"
30         android:layout_width="200dp"
31         android:layout_height="200dp"
32         android:src="@mipmap/ic_launcher"/>
33
34 </LinearLayout>
```

ModeloPersona

Creamos una clase java para caracterizar el comportamiento de un objeto de tipo persona a la que llamaremos "**ModeloPersona**".

```
1 package com.example.dell.proyecto006_listviewpersona;
2 public class ModeloPersona {
3     private String nombre;
4     private String descripcion;
5     private int codigo;
6     private int foto;
7
8     public String getNombre() {
9         return nombre;
10    }
11
12    public void setNombre(String nombre){
13        this.nombre=nombre;
14    }
15
16    public String getDescripcion() {
17        return descripcion;
18    }
19
20    public void setDescripcion(String descripcion) {
21        this.descripcion = descripcion;
22    }
23
24    public int getCodigo() {
25        return codigo;
26    }
27
28    public void setCodigo(int codigo) {
29        this.codigo = codigo;
30    }
```

```
31     public int getFoto() {
32         return foto;
33     }
34
35     public void setFoto(int foto) {
36         this.foto = foto;
37     }
38 }
39 }
```

MainActivity.java

```
1 package com.example.dell.proyecto006_listviewpersona;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.util.Log;
6 import android.view.View;
7 import android.widget.AdapterView;
8 import android.widget.ListView;
9 import android.widget.Toast;
10 import java.util.ArrayList;
11
12 public class MainActivity extends AppCompatActivity {
13     private ListView lv;
14     private Adaptador adaptador;
15
16     @Override
17
18     protected void onCreate(Bundle savedInstanceState) {
19         super.onCreate(savedInstanceState);
20         setContentView(R.layout.activity_main);
```

```
21     ArrayList <ModeloPersona> personas = new ArrayList<ModeloPersona>();
22
23     ModeloPersona m1=new ModeloPersona();
24     m1.setNombre("Pepe");
25     m1.setDescripcion("Tranquilo y con ganas de vivir");
26     m1.setCodigo(1);
27     m1.setFoto(R.drawable.foto1);
28     ModeloPersona m2=new ModeloPersona();
29     m2.setNombre("Rosa");
30     m2.setDescripcion("Ambiciosa y sin complejos");
31     m2.setCodigo(2);
32     m2.setFoto(R.drawable.foto2);
33     ModeloPersona m3=new ModeloPersona();
34     m3.setNombre("Chiky");
35     m3.setDescripcion("Capaz de todo por amor");
36     m3.setCodigo(3);
37     m3.setFoto(R.drawable.foto3);
38     ModeloPersona m4=new ModeloPersona();
39     m4.setNombre("Piluca");
40     m4.setDescripcion("Herida pero dispuesta a perdonar");
41     m4.setCodigo(4);
42     m4.setFoto(R.drawable.foto4);
43
44     personas.add(m1);
45     personas.add(m2);
46     personas.add(m3);
47     personas.add(m4);
48     adaptador=new Adaptador(this, personas);
49
50     lv = (ListView)findById(R.id.lv);
51     lv.setAdapter(adaptador);
52
53     lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
54
55         @Override
56
57         public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
58             try{
59                 ModeloPersona persona=(ModeloPersona)adaptador.getItem(i);
```

```
60         Log.e("persona", persona.getCodigo()+"-"+persona.getNombre());
61         Toast.makeText(getApplicationContext(), "El código es: "+persona.getCodigo(), Toast.LENGTH_LONG)
62     } catch (Exception e){
63         e.printStackTrace();
64     }
65 }
66 });
67 }
68 }
```



Adaptador.java

Aquí va nuestra clase de tipo adaptador.

```
1 import java.util.ArrayList;
2
3 public class Adaptador extends BaseAdapter {
4     private Context miContexto;
5     private ArrayList <ModeloPersona> miArrayList;
6
7     public Adaptador(Context miContexto, ArrayList<ModeloPersona> miArrayList){
8         this.miContexto=miContexto;
9         this.miArrayList=miArrayList;
10    }
11
12    @Override
13
14    public int getCount() {
15        return miArrayList.size();
16    }
17}
```

```
18     @Override
19
20     public Object getItem(int i) {
21         return miArrayList.get(i);
22     }
23
24     @Override
25
26     public long getItemId(int i) {
27         return miArrayList.get(i).getCodigo();
28     }
29
30     @Override
31
32     public View getView(int i, View view, ViewGroup viewGroup) {
33         LayoutInflator layoutInflater=LayoutInflater.from(miContexto);
34         view = layoutInflater.inflate(R.layout.item, null);
35         TextView nombre=(TextView)view.findViewById(R.id.tvNombre);
36         TextView descripcion=(TextView)view.findViewById(R.id.tvDescripcion);
37         ImageView foto=(ImageView)view.findViewById(R.id.imgFoto);
38         nombre.setText(miArrayList.get(i).getNombre());
39         descripcion.setText((miArrayList.get(i).getDescripcion()));
40         foto.setImageResource(miArrayList.get(i).getFoto());
41         return view;
42     }
43 }
```

4.- Menús.

Caso práctico



Mikhail (Pexels)

- Ahora -dice Juan- empieza a echar en falta los menús. Hacen que sea más fácil interactuar con los usuarios.
- Vamos a ello -anima Ada-. Normalmente nos encontramos con aplicaciones que usan tres tipos de menús: **menú de opciones** principales de la app o activity, normalmente accesible desde la propia barra de la aplicación; **menú contextual**, desplegado encima de un elemento cuando el usuario hace una pulsación larga sobre el mismo; **menú emergente**, desplegado en la aplicación en general.

Los menús serán componentes habituales en aplicaciones Android. Nos encontraremos estas tres posibles situaciones en relación al uso de menús:

✓ Menú de opciones y barra de app.

El menú de opciones es la colección principal de elementos de menú para una actividad. Es donde debemos colocar las acciones que tienen un impacto global en la app, como “Buscar”, “Enviar un mensaje” y “Configuración”.

✓ Menú contextual y modo de acción contextual.

Un menú contextual es un menú flotante que aparece cuando el usuario hace un clic largo en un elemento (mantiene pulsado un elemento durante aproximadamente un segundo). Proporciona acciones que afectan el contenido seleccionado o el marco contextual.

En el menú de acción contextual se muestran los elementos de acción que afectan el contenido seleccionado en una barra en la parte superior de la pantalla y se permite al usuario seleccionar varios elementos.

Menú emergente.

Un menú emergente muestra una lista de elementos en una lista vertical que está anclada a la vista que invocó el menú. Es adecuado para proporcionar una ampliación de acciones relacionadas con contenido específico o para proporcionar opciones para una segunda parte de un comando. Las acciones en un menú emergente **no** deben afectar directamente el contenido correspondiente, para eso están las acciones contextuales. En cambio, el menú emergente es para acciones extendidas relacionadas con partes del contenido de la actividad.

4.1.- Definición de un menú en XML.

Para todos los tipos de menús, Android proporciona un formato XML estándar para definir los elementos del menú. En lugar de incorporar un menú en el código de la actividad, debes definir un menú y todos los elementos en un recurso de menú XML (se encontrará en **res/menu/fichero_menu.xml**). Luego, podremos utilizar dicho recurso de menú (cargarlo como un objeto **Menu**) en la actividad o el fragmento.

El uso del recurso de menú es una práctica recomendada por algunos motivos:

- ✓ Es más fácil visualizar la estructura del menú en XML.
- ✓ Separa el contenido del menú del código de comportamiento de la aplicación.
- ✓ Nos permite crear configuraciones alternativas del menú para diferentes versiones de la plataforma, tamaños de pantallas y otras configuraciones aprovechando el framework de recursos de la app.

Para definir el menú, crearemos un archivo XML dentro del directorio **res/menu/** del proyecto y desarrollaremos el menú con los siguientes elementos:

- ✓ **<menu>** Define un objeto **Menu**, que es un contenedor para elementos del menú. Un elemento **<menu>** debe ser el nodo raíz del archivo y puede tener uno o más elementos **<item>** y **<group>**.
- ✓ **<item>** Crea un **MenuItem**, que representa un único elemento en un menú. Este elemento puede contener un elemento **<menu>** anidado para crear un submenú.
- ✓ **<group>** Contenedor opcional e invisible para elementos **<item>**. Nos permite categorizar los elementos del menú para que compartan propiedades, como el estado de una actividad o visibilidad.

Ejemplo

Presentamos un menú de ejemplo denominado **game_menu.xml**:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3     <item android:id="@+id/new_game"
```

```
4     android:icon="@drawable/ic_new_game"
5     android:title="@string/new_game"
6     android:showAsAction="ifRoom"/>
7
8     <item android:id="@+id/help"
9         android:icon="@drawable/ic_help"
10        android:title="@string/help" />
11
12 </menu>
```

El elemento **<item>** admite varios atributos que puedes usar para definir la apariencia y el comportamiento de un elemento. Los elementos del menú precedente incluyen estos atributos (estos son los más importantes, pero hay más):

- ✓ **android:id:** ID de un recurso que es exclusivo del elemento y permite que la aplicación lo reconozca cuando el usuario lo selecciona.
- ✓ **android:icon:** Referencia a un elemento de diseño para usar como el ícono del elemento.
- ✓ **android:title:** Referencia a una string para usar como el título del elemento.
- ✓ **android:showAsAction:** Especifica cuándo y cómo el elemento debe aparecer como un elemento de acción en la barra de app.

Podemos agregar un submenú a un elemento en cualquier menú (salvo en un submenú) agregando un elemento **<menu>** como campo secundario de un **<item>**. Los submenús son útiles cuando la aplicación tiene muchas funciones que se pueden organizar en temas, como elementos de una barra de menús de una aplicación para escritorio (Archivo, Editar, Ver, etc.).

Ejemplo

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3     <item android:id="@+id/file"
4         android:title="@string/file" >
5         <!-- "file" submenu -->
6         <menu>
```

```
7      <item android:id="@+id/create_new"
8          android:title="@string/create_new" />
9      <item android:id="@+id/open"
10         android:title="@string/open" />
11    </menu>
12  </item>
13 </menu>
```

Para usar un submenú en la actividad, tendremos que convertir el recurso XML en un objeto programable con **MenuInflater.inflate()**.

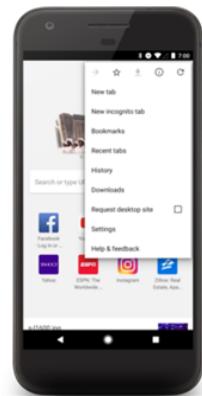
4.2.- Creación de menú de opciones.

El menú de opciones es dónde debes incluir las acciones y otras opciones que son relevantes para el contexto de la actividad actual, como "Buscar", "Redactar correo electrónico" y "Ajustes".

Que los elementos de las opciones aparezcan en el menú en la pantalla depende de la versión para la que desarrollemos la aplicación:

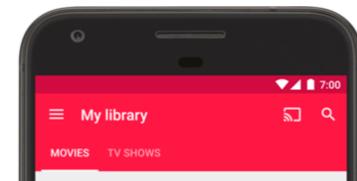
- ✓ Si desarrollamos la aplicación en **Android 2.3.x (nivel de API 10) o versiones anteriores**, el contenido del menú de opciones aparece en la parte inferior de la pantalla cuando el usuario presiona el botón *Menú*, como se muestra en la figura 1. Cuando se abre, la primera parte visible es el menú de íconos, que tiene hasta seis elementos de menú. Si el menú incluye más de seis elementos, Android coloca el sexto elemento en adelante en el menú ampliado, que se puede abrir seleccionando "**Más**".
- ✓ Si desarrollamos la aplicación en **Android 3.0 (nivel de API 11) y versiones posteriores**, los elementos del menú de opciones están disponibles en la barra de app. De forma predeterminada, el sistema dispone todos los elementos en acciones adicionales, que el usuario puede ver con el ícono de acciones adicionales a la derecha de la barra de app (o mediante el botón *Menú* del dispositivo si está disponible). Para permitir el acceso rápido a acciones importantes, podemos hacer que algunos elementos aparezcan en la barra de app agregando **android:showAsAction="ifRoom"** a los elementos **<item>** correspondientes (consulta la figura 2).

Figura 1



[Google Developers](#) (Uso educativo nc)

Figura 2



[Google Developers](#) (Uso educativo nc)

Podemos declarar elementos para el menú de opciones desde una **Activity** o desde un **Fragment**. Si tanto la actividad como los fragmentos declaran elementos para el menú de opciones, se los combina en la IU. Los elementos de la actividad aparecen primero y

después los elementos de cada fragmento, en el orden en que cada fragmento se agregó a la actividad. Si es necesario, puedes reordenar los elementos del menú con el atributo android:orderInCategory en cada **<item>** que necesites mover.

Para especificar el menú de opciones para una actividad, debemos sobrescribir el método **onCreateOptionsMenu()** (los fragmentos proporcionan su propio callback **onCreateOptionsMenu()**). En este método, podremos recoger el recurso de menú (definido en XML).

Ejemplo

```
1  @Override  
2  
3  public boolean onCreateOptionsMenu(Menu menu) {  
4      MenuInflater inflater = getMenuInflater();  
5      inflater.inflate(R.menu.game_menu, menu);  
6      return true;  
7 }
```

También podemos agregar elementos del menú con **add()** y recuperar elementos con **findItem()**.

Si hemos desarrollado la aplicación para Android 2.3.x y versiones anteriores, el sistema llama a **onCreateOptionsMenu()** para crear el menú de opciones cuando el usuario abre el menú por primera vez. Si la desarrollaste para Android 3.0 y versiones posteriores, el sistema llama a **onCreateOptionsMenu()** cuando comienza la actividad, para mostrar los elementos de la barra de app.

4.3.- Manejo de eventos de clic.

Cuando el usuario selecciona un elemento del menú de opciones (incluidos los elementos de acción de la barra de app), el sistema llama al método **onOptionsItemSelected()** de la actividad. Este método pasa el **MenuItem** seleccionado. Podremos identificar el elemento si llamamos a **getItemId()**, que muestra el ID único del elemento del menú (definido por el atributo **android:id** en el recurso de menú o con un valor entero proporcionado al método **add()**). Podemos hacer coincidir este ID con elementos del menú conocidos para realizar la acción correspondiente.

Ejemplo

```
1  @Override
2
3  public boolean onOptionsItemSelected(MenuItem item) {
4      // Handle item selection
5      switch (item.getItemId()) {
6          case R.id.new_game:
7              newGame();
8              return true;
9          case R.id.help:
10             showHelp();
11             return true;
12         default:
13             return super.onOptionsItemSelected(item);
14     }
15 }
```

Cuando controlemos correctamente un elemento del menú, mostraremos true. Si no controlas el elemento del menú, debes llamar a la implementación de la superclase de **onOptionsItemSelected()** (la implementación predeterminada muestra false).

Si la actividad incluye fragmentos, el sistema llama primero a **onOptionsItemSelected()** para la actividad y, a continuación, para cada fragmento (en el orden en que se agregaron) hasta que uno muestre true o se llamen todos los fragmentos.

Sugerencia: Desde Android 3.0 se incluye la capacidad de definir el comportamiento haciendo clic en un elemento del menú en XML, mediante el atributo android:onClick. El valor del atributo debe ser el nombre de un método definido por la actividad que usa el menú. El método debe ser público y aceptar un único parámetro **MenuItem**, cuando el sistema llama a este método, pasa el elemento del menú seleccionado.

Sugerencia: Si la aplicación contiene varias actividades y algunas de ellas proporcionan el mismo menú de opciones, debemos considerar crear una actividad que solo implemente los métodos **onCreateOptionsMenu()** y **onOptionsItemSelected()**. Luego, extenderemos esta clase por cada actividad que deba compartir el mismo menú de opciones. De esta manera, podemos administrar un conjunto de código para manejar las acciones del menú y cada clase descendiente hereda el comportamiento del menú. Si deseamos agregar elementos del menú a una de las actividades descendientes, invalidaremos sobrescribiendo **onCreateOptionsMenu()** en esa actividad. Llamaremos a **super.onCreateOptionsMenu(menu)** para que se creen los elementos originales del menú y luego agrega elementos nuevos con **menu.add()**. También puedes invalidar el comportamiento de la superclase para elementos del menú individuales.

4.4.- Creación de menús contextuales.



[Google Developers](#) (Uso educativo
nc)

Un menú contextual ofrece acciones que afectan un elemento o marco contextual específicos en la IU. Puedes proporcionar un menú contextual para cualquier vista, aunque estos se usan con mayor frecuencia para elementos en **ListView**, **GridView** u otras colecciones de vistas en las cuales el usuario puede realizar acciones directamente en cada elemento.

En un menú contextual flotante. Un menú aparece como una lista flotante de elementos del menú (similar a un cuadro de diálogo) cuando el usuario realiza un clic largo (presiona y mantiene presionado) en una vista que declara admitir un menú contextual. Los usuarios pueden realizar una acción contextual en un elemento a la vez.

4.5.- Creación de un menú contextual flotante.

Para crear un menú contextual flotante:

1. Registraremos la **View** con la que se debe asociar el menú contextual llamando a **registerForContextMenu()** y pasándolo a la **View**.

Si la actividad usa un **ListView** o **GridView**, y deseamos que cada elemento proporcione el mismo menú contextual, registraremos todos los elementos de un menú contextual pasando **ListView** o **GridView** a **registerForContextMenu()**.

2. Implementaremos el método **onCreateContextMenu()** en nuestra **Activity** o **Fragment**.

Cuando la vista registrada recibe un evento de clic largo, el sistema llama al método **onCreateContextMenu()**. Aquí es donde definimos los elementos del menú, por lo general agregando un recurso de menú.

Ejemplo

```
1  @Override  
2  
3  public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuItemInfo menuInfo) {  
4      super.onCreateContextMenu(menu, v, menuInfo);  
5      MenuInflater inflater = getMenuInflater();  
6      inflater.inflate(R.menu.context_menu, menu);  
7 }
```

MenuInflater permite agregar el menú contextual desde un recurso de menú. Los parámetros del método callback incluye la **View** que el usuario seleccionó y el objeto **ContextMenu.ContextMenuItemInfo** que proporciona información adicional sobre el elemento seleccionado. Si la actividad tiene varias vistas, cada una de las cuales proporciona un menú contextual diferente, podremos usar estos parámetros para determinar qué menú contextual debemos agregar.

3. Implementaremos **onContextItemSelected()**.

Cuando un usuario selecciona un elemento del menú, el sistema llama a este método para que podamos realizar la acción apropiada.

Ejemplo

```
1  @Override
2
3  public boolean onContextItemSelected(MenuItem item) {
4      AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
5      switch (item.getItemId()) {
6          case R.id.edit:
7              editNote(info.id);
8              return true;
9          case R.id.delete:
10             deleteNote(info.id);
11             return true;
12         default:
13             return super.onContextItemSelected(item);
14     }
15 }
```

El método **getItemId()** consulta el ID del elemento del menú seleccionado, que debemos asignar a cada elemento del menú en XML con el atributo **android:id**.

4.6.- Creación de un menú emergente.

Un **PopupMenu** es un menú modal anclado a una **View**. Aparece debajo de la vista anclada si hay espacio o, de lo contrario, sobre esta.

Si definimos el menú en XML, podemos hacer esto para mostrar el menú emergente:

1. Crearemos una instancia de **PopupMenu** con su constructor, que toma el **Context** de la aplicación actual y la **View** a la cual el menú debe anclarse.
2. Usaremos **MenuInflater** para inflar el recurso de menú hacia el objeto **Menu** devuelto por **PopupMenu.getMenu()**.
3. Llamaremos a **PopupMenu.show()**

Ejemplo

Aquí se presenta un botón con el atributo **android:onClick** que muestra un menú emergente:

```
1 <ImageButton  
2     android:layout_width="wrap_content"  
3     android:layout_height="wrap_content"  
4     android:src="@drawable/ic_overflow_holo_dark"  
5     android:contentDescription="@string/descr_overflow_button"  
6     android:onClick="showPopup" />
```

La actividad puede mostrar el menú emergente de la siguiente manera:

```
1 public void showPopup(View v) {  
2     PopupMenu popup = new PopupMenu(this, v);  
3     MenuInflater inflater = popup.getMenuInflater();  
4     inflater.inflate(R.menu.actions, popup.getMenu());  
5 }
```

```
6 |     popup.show();
| }
```

En el nivel de API 14 y en niveles superiores, podemos combinar las dos líneas que agrandan el menú con **PopupMenu.inflate()**.

El menú se cierra cuando el usuario selecciona un elemento o toca fuera del área del menú. Puedes recibir el evento de cierre con **PopupMenu.OnDismissListener**.

4.7.- Manejo de eventos de clic.

Para realizar una acción cuando el usuario selecciona un elemento del menú, debes implementar la interfaz de **PopupMenu.OnMenuItemClickListener** y registrarla en el **PopupMenu** llamando a **setOnMenuItemClickListener()**. Cuando el usuario selecciona un elemento, el sistema llama al callback **onMenuItemClick()** en la interfaz.

Ejemplo

```
1 public void showMenu(View v) {  
2     PopupMenu popup = new PopupMenu(this, v);  
3     // This activity implements OnMenuItemClickListener  
4     popup.setOnMenuItemClickListener(this);  
5     popup.inflate(R.menu.actions);  
6     popup.show();  
7 }  
8  
9 @Override  
10  
11 public boolean onMenuItemClick(MenuItem item) {  
12     switch (item.getItemId()) {  
13         case R.id.archive:  
14             archive(item);  
15             return true;  
16         case R.id.delete:  
17             delete(item);  
18             return true;  
19         default:  
20             return false;  
21     }  
22 }
```


4.8.- Ejercicio resuelto 6: Items.

Ejercicio Resuelto

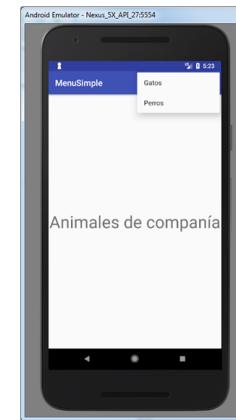
Vamos a tratar de crear un menú sobre la ActionBar de nuestra aplicación, de forma que aparezcan un par de opciones y que estas sean clicables. En nuestro caso simplemente abrirán una página web asociada al ítem clicado.

La apariencia inicial:



[Google Developers \(Uso educativo nc\)](#)

Al clicar en el menú, se desplegarán las opciones:



[Google Developers \(Uso educativo nc\)](#)

Al clicar en cada uno de los ítems de opción, se abrirá un página web:



[Google Developers](#) (Uso educativo nc)



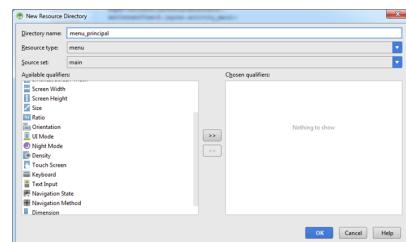
[Google Developers](#) (Uso educativo nc)

Mostrar retroalimentación

Crearemos un proyecto nuevo, y dentro de la carpeta de recursos **res**, crearemos un nuevo directorio de recursos que llamaremos **menu** (figura 1).

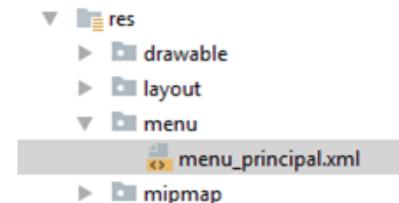
A su vez, dentro de dicha carpeta crearemos un fichero de recursos de Menú (**Menu Resources File**), que en mi caso llamaré **menu_principal.xml** (figura 2).

Figura 1



[Google Developers](#) (Uso educativo nc)

Figura 2



[Google Developers](#) (Uso educativo nc)

menu_principal.xml

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3     <item
4         android:id="@+id/itemGatos"
5         android:title="Gatos" />
6     <item
7         android:id="@+id/itemPerros"
8         android:title="Perros" />
9 </menu>
```

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <android.support.constraint.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     tools:context="com.example.dell.menusimple.MainActivity">
9
10    <TextView
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:text="Animales de compañía"
```

```
14     android:textSize="40dp"
15     android:gravity="center"
16     app:layout_constraintBottom_toBottomOf="parent"
17     app:layout_constraintLeft_toLeftOf="parent"
18     app:layout_constraintRight_toRightOf="parent"
19     app:layout_constraintTop_toTopOf="parent" />
20
21 </android.support.constraint.ConstraintLayout>
```

MainActivity.java

```
1 package com.example.dell.menusimple;
2
3 import android.content.Intent;
4 import android.net.Uri;
5 import android.support.v7.app.AppCompatActivity;
6 import android.os.Bundle;
7 import android.view.Menu;
8 import android.view.MenuInflater;
9 import android.view.MenuItem;
10
11 public class MainActivity extends AppCompatActivity {
12
13     @Override
14
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18     }
19
20     @Override
```

```
22
23     public boolean onCreateOptionsMenu(Menu menu) {
24         MenuInflater inflater=getMenuInflater();
25         inflater.inflate(R.menu.menu_principal, menu);
26         return true;
27     }
28
29     @Override
30
31     public boolean onOptionsItemSelected(MenuItem item) {
32         Intent i;
33         switch (item.getItemId()) {
34             case R.id.itemGatos: i = new Intent("android.intent.action.VIEW", Uri.parse("https://www.anipedia
35                         startActivity(i);
36                         break;
37             case R.id.itemPerros: i = new Intent("android.intent.action.VIEW", Uri.parse("https://www.anipedi
38                         startActivity(i);
39                         break;
40         }
41         return true;
42     }
43 }
```



AndroidManifest.xml

No debemos olvidar dar permisos de acceso a Internet en el **AndroidManifest.xml**:

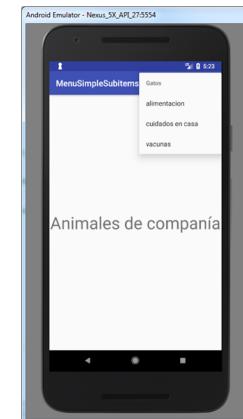
```
1 | <uses-permission android:name="android.permission.INTERNET"/>
```



4.9.- Ejercicio resuelto 7: SubItems.

Ejercicio Resuelto

Vamos a incorporar para el ejercicio anterior, en alguno de los ítems, subítems. Por ejemplo al pulsar el ítem Gatos, aparecerán sus ítems correspondientes.



[Google Developers](#) (Uso educativo nc)

Mostrar retroalimentación

menu_principal.xml

Copiamos el proyecto anterior y modificamos el fichero **menu_principal.xml** del menú de la siguiente forma:

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3     <item
4         android:id="@+id/itemGatos"
5         android:title="Gatos" >
6         <menu>
7             <item
8                 android:id="@+id/itemGatos1"
9                 android:title="alimentacion"/>
10            <item
11                android:id="@+id/itemGatos2"
12                android:title="cuidados en casa"/>
13            <item
14                android:id="@+id/itemGatos3"
15                android:title="vacunas"/>
16        </menu>
17    </item>
18    <item
19        android:id="@+id/itemPerros"
20        android:title="Perros" />
21 </menu>
```

MainActivity.java

...y por otro lado **MainActivity.java**:

```
1 package com.example.dell.menusimplesubitems;
2
3 import android.content.Intent;
4 import android.net.Uri;
5 import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
6 import android.view.Menu;
7 import android.view.MenuInflater;
8 import android.view.MenuItem;
9
10 public class MainActivity extends AppCompatActivity {
11
12     @Override
13
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17     }
18
19     @Override
20     public boolean onCreateOptionsMenu(Menu menu) {
21         MenuInflater inflater=getMenuInflater();
22         inflater.inflate(R.menu.menu_principal, menu);
23         return true;
24     }
25
26     @Override
27
28     public boolean onOptionsItemSelected(MenuItem item) {
29         Intent i;
30         switch (item.getItemId()) {
31             case R.id.itemGatos1: i = new Intent("android.intent.action.VIEW", Uri.parse("https://es.wikipedia.org/wiki/Gato"));
32                 startActivity(i);
33                 break;
34             case R.id.itemGatos2: i = new Intent("android.intent.action.VIEW", Uri.parse("https://www.tiendanube.com/search?query=gato"));
35                 startActivity(i);
36                 break;
37             case R.id.itemGatos3: i = new Intent("android.intent.action.VIEW", Uri.parse("https://www.tiendanube.com/search?query=gato"));
38                 startActivity(i);
39                 break;
40             case R.id.itemPerros: i = new Intent("android.intent.action.VIEW", Uri.parse("https://www.anipedia.net/search?query=perro"));
41                 startActivity(i);
42                 break;
43         }
44         return true;
45     }
46 }
```

```
45 }  
46 }  
47 }
```

4.10.- Ejercicio resuelto 8: Menú contextual.

Ejercicio Resuelto

En este caso, vamos a hacer un ejercicio donde trabajemos con menús contextuales. Concretamente, mi menú contextual aparecerá al hacer clic sobre un **TextView**, aparecerá un menú emergente que permitirá cambiar el color de fondo de un layout interno.

La apariencia inicial, será:



[Google Developers \(Uso educativo nc\)](#)

Al mantener pulsado sobre el texto, se lanzará el menú emergente:



[Google Developers](#) (Uso educativo nc)

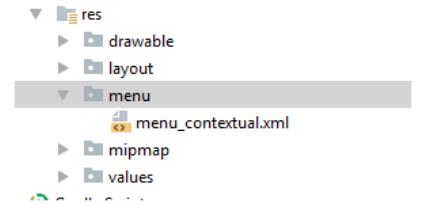
Si elijo, por ejemplo "Color rojo":



[Google Developers](#) (Uso educativo nc)

Mostrar retroalimentación

Crearemos un fichero **menu_contextual.xml**, dentro de la carpeta de recursos menu.
Su contenido será **menu_contextual.xml**.



[Google Developers](#) (Uso educativo nc)

menu_contextual.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:app="http://schemas.android.com/apk/res-auto"
3     xmlns:android="http://schemas.android.com/apk/res/android">
4     <item
5         android:id="@+id/itemRojo"
6         android:title="Color rojo" />
7     <item
8         android:id="@+id/itemVerde"
9         android:title="Color verde" />
10    <item
11        android:id="@+id/itemAzul"
12        android:title="Color azul" />
13    </menu>
```

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical"
8     tools:context="com.example.dell.menucontextualbasico.MainActivity">
9
10    <TextView
11        android:id="@+id/tv1"
12        android:layout_width="match_parent"
13        android:layout_height="wrap_content"
14        android:text="Mantenme pulsado para cambiar color de fondo"
15        android:gravity="center"
16        android:textSize="25dp"
17        android:background="#FFFFFF"/>
18
19    <LinearLayout
20        android:id="@+id/layoutInterno"
21        android:layout_width="match_parent"
22        android:layout_height="match_parent"
23        android:orientation="vertical">
24    </LinearLayout>
25
26 </LinearLayout>
```

MainActivity.java

```
1 package com.example.dell.menucontextualbasico;
2 import android.graphics.Color;
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.text.Layout;
6 import android.view.ContextMenu;
7 import android.view.MenuInflater;
8 import android.view.MenuItem;
9 import android.view.View;
10 import android.widget.EditText;
11 import android.widget.LinearLayout;
12 import android.widget.TextView;
13
14 public class MainActivity extends AppCompatActivity {
15     TextView tv1;
16     LinearLayout layoutInterno;
17
18     @Override
19
20     public void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_main);
23         layoutInterno=(LinearLayout)findViewById(R.id.layoutInterno);
24         tv1=(TextView)findViewById(R.id.tv1);
25         registerForContextMenu(tv1);
26     }
27
28     @Override
29
30     public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)
31     {
32         menu.setHeaderTitle("Elija el color de fondo:");
33         MenuInflater inflater = getMenuInflater();
34         inflater.inflate(R.menu.menu_contextual, menu);
35     }
36
```

```
37     @Override
38
39     public boolean onContextItemSelected(MenuItem item) {
40         switch (item.getItemId()) {
41             case R.id.itemRojo:layoutInterno.setBackgroundColor(Color.rgb(255, 0, 0)) ;
42                 break;
43             case R.id.itemVerde:layoutInterno.setBackgroundColor(Color.rgb(0, 255, 0)) ;
44                 break;
45             case R.id.itemAzul:layoutInterno.setBackgroundColor(Color.rgb(0, 0, 255)) ;
46                 break;
47         }
48         return true;
49     }
50 }
```